THE UNIVERSITY OF TEXAS AT DALLAS

Hadoop Setup

Erick Skorupa Parolin

Erik Jonsson School of Engineering and Computer Science

Learning Outcomes

In this tutorial, we will cover the following steps:

- Install Virtual Machine on your computers
- Java installation on VM
- Hadoop installation on VM
- Install & Configure IDE for running Hadoop job
- Run our first Hadoop code (WordCount)

THE UNIVERSITY OF TEXAS AT DALLAS

Virtual Machine

Installation Guide

Learning Outcomes

Upon completion of this lesson, students will be able to:

- Install Virtual Machine on their computers

  Note: This step is required only for those who use Windows or other non-POSIX OS

OS INFO

- Linux is well supported by almost all big data frameworks.
- Mac (Unix-like) is similar to Linux and should be fine for most frameworks.
- Windows is not well supported and may require extra effort to make it work.
- However, building your virtual machine with Linux as your OS is highly recommended for all of you.
- Never try to run codes on windows in this course. It is not fully compatible and hard to debug!

Installation of VIRTUAL Machine

- Please always test your code on your machine before try to submit and run it on a server.
  - The Server might shut down due to unexpected accidents. Running code on your own machine alleviates the effort of debugging.
  - Make sure your machine meet following requirements:
    >=8G RAM (GUI-supported Linux only)
    >=4G RAM (for command-line Linux only)
    >=16G RAM is highly recommended for running big data framework without significant delay

Linux info

- In this course, we choose Ubuntu as desired Linux distribution.
- If you are familiar with Linux, CentOS is also a good choice.
- Linux info:
  - Ubuntu 20.04.2 LTS (my setup: 18.04.3 LTS)
  - https://www.ubuntu.com/download/desktop
  - Click above url and click on download to download Ubuntu disk image
- VirtualBox
  - https://www.virtualbox.org/wiki/Downloads
  - Click above url to download VirtualBox

VirtualBox

- Click on url corresponding to your operating system
- We take mac os as example
- Double click on the downloaded file (e.g. VirtualBox-6.1.18-142142-Win.exe) to install

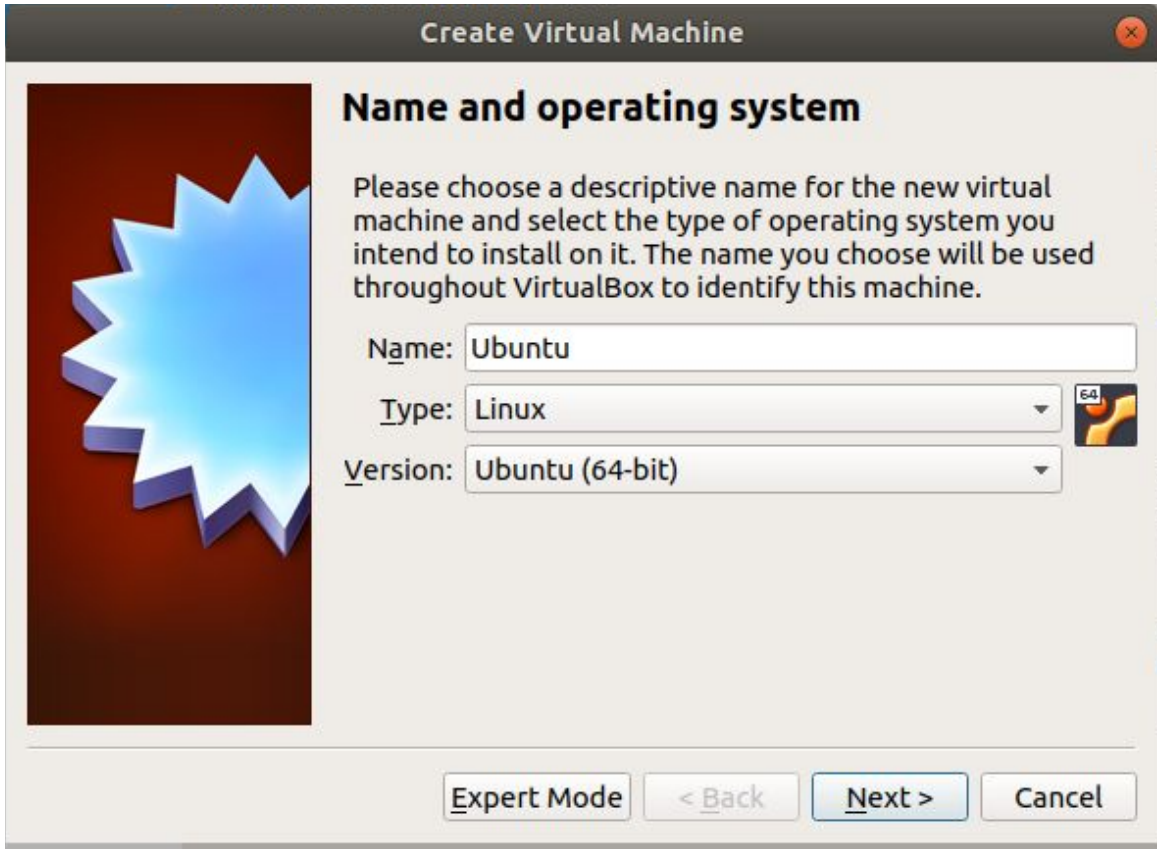
Installation

- Open VirtualBox

Installation

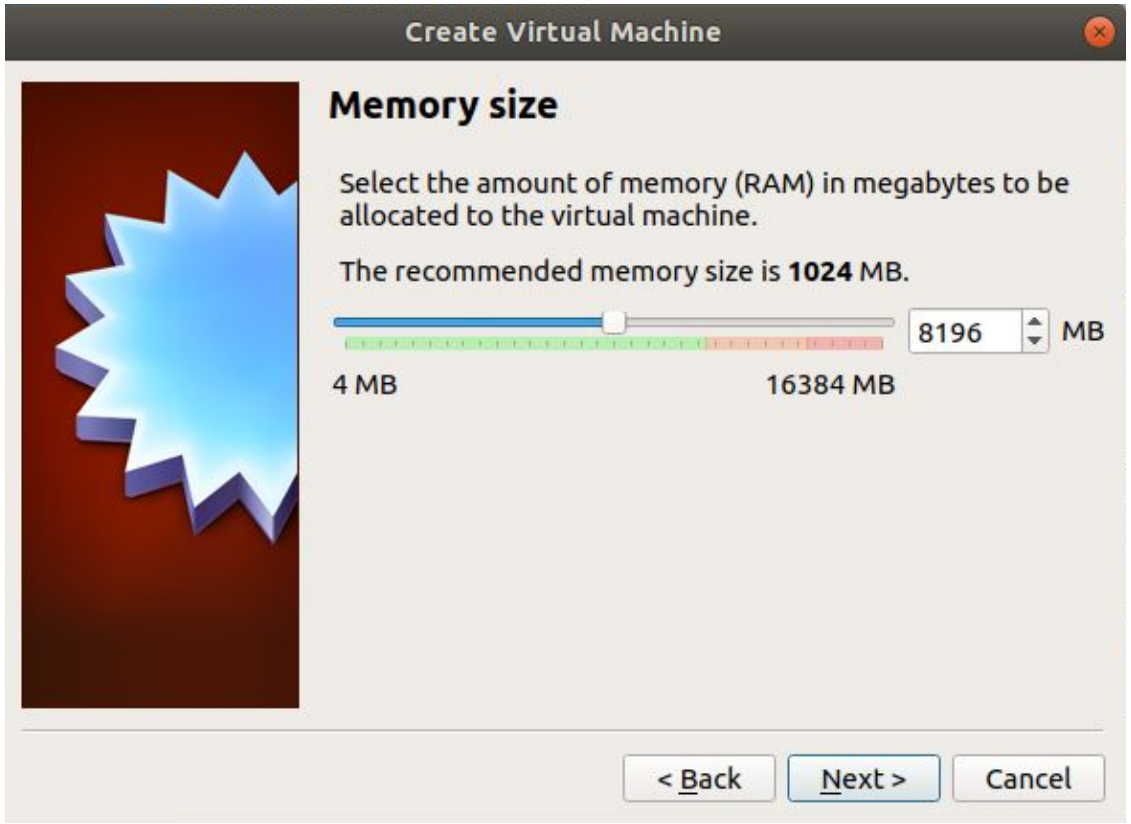
- Click on 

- Enter Ubuntu
- Click on Continue



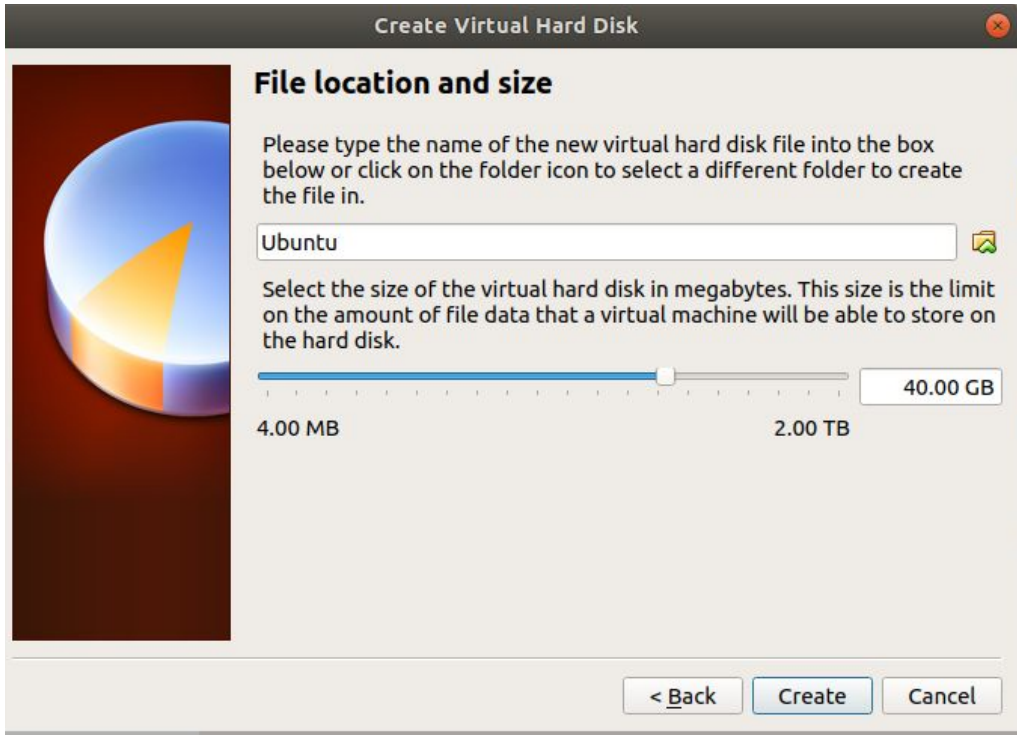
Installation

- Set memory size to be either 4096 or 8192 MB
- Here we choose 8192 MB for better performance.
- Click on Next
- Click on **Create** to "*Create a virtual hard disk now*"
- Keep "*VDI*" and click on Next
- Keep "*Dynamically allocated*" and click Next



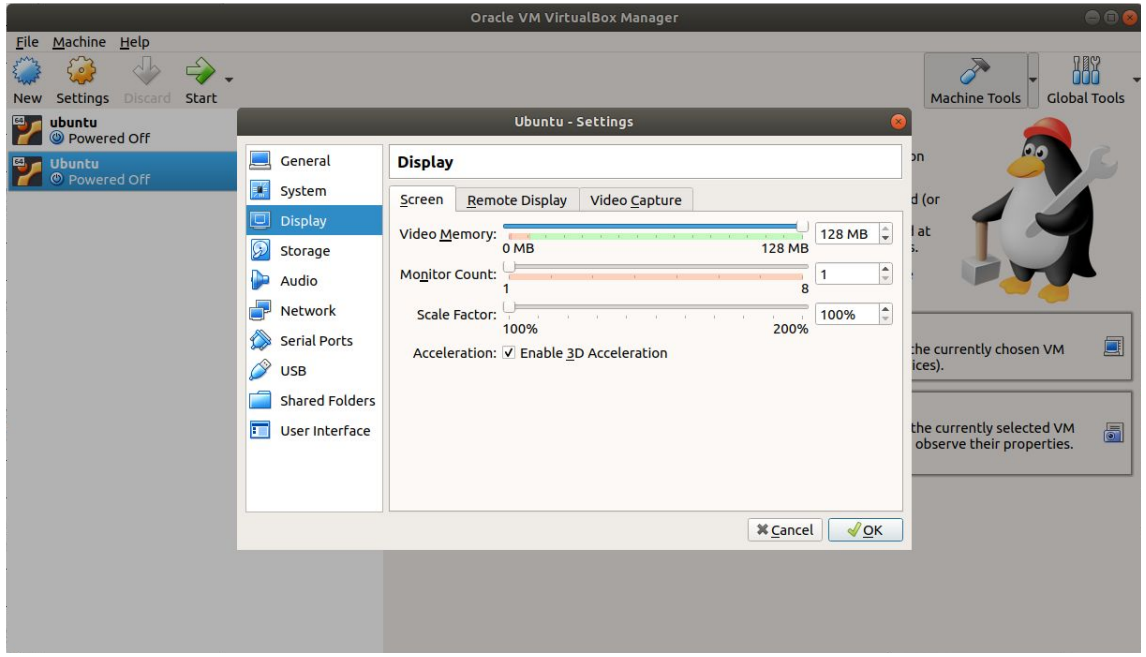
Installation

- Setup disk size you prefer (**at least** 40 GB)
- Click on Create



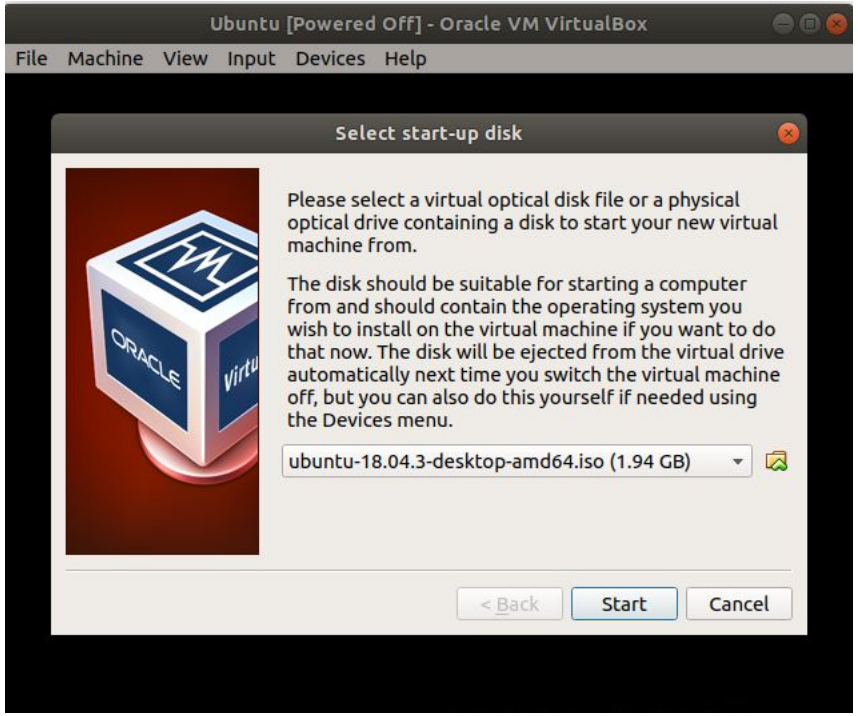
Installation

- You will see the VM you just created in the VirtualBox Manager
- Select your VM, click on Settings and then Display
- Increase Video Memory to 128 MB
- Check Enable 3D Acceleration
- Click on OK
- Click on Start to start virtual machine



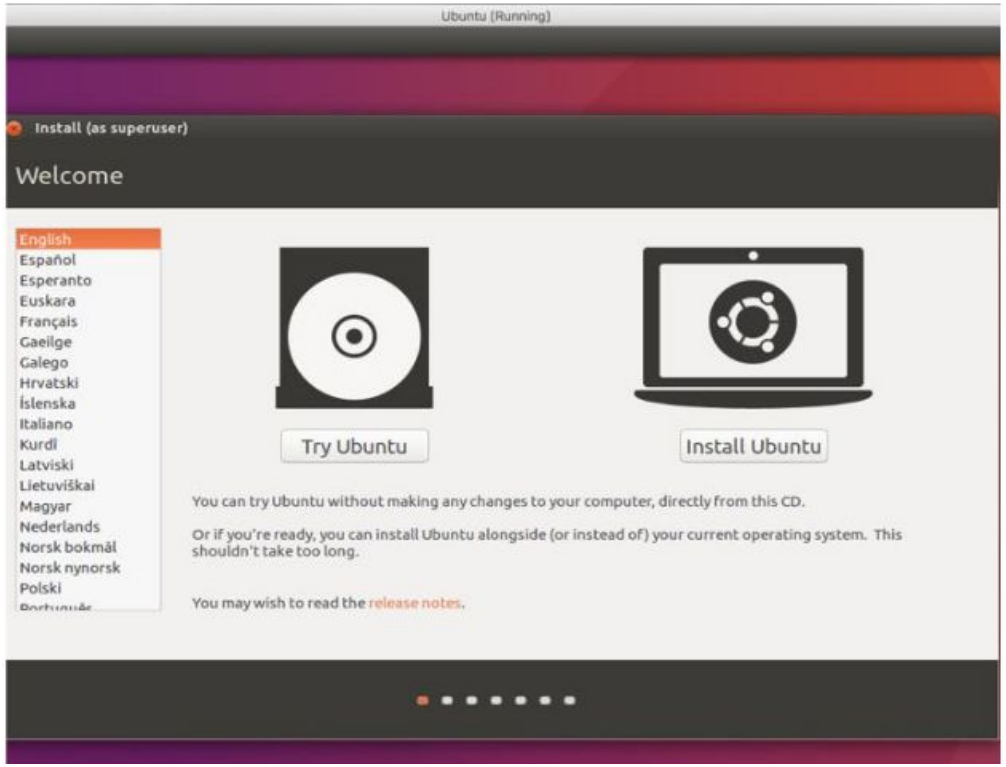
Installation

- A window “Select start-up disk” is prompt out.
- Select the Ubuntu disk image you just download.
- Click on Start



Installation

- You are going to see a window like right image
- Click on install Ubuntu

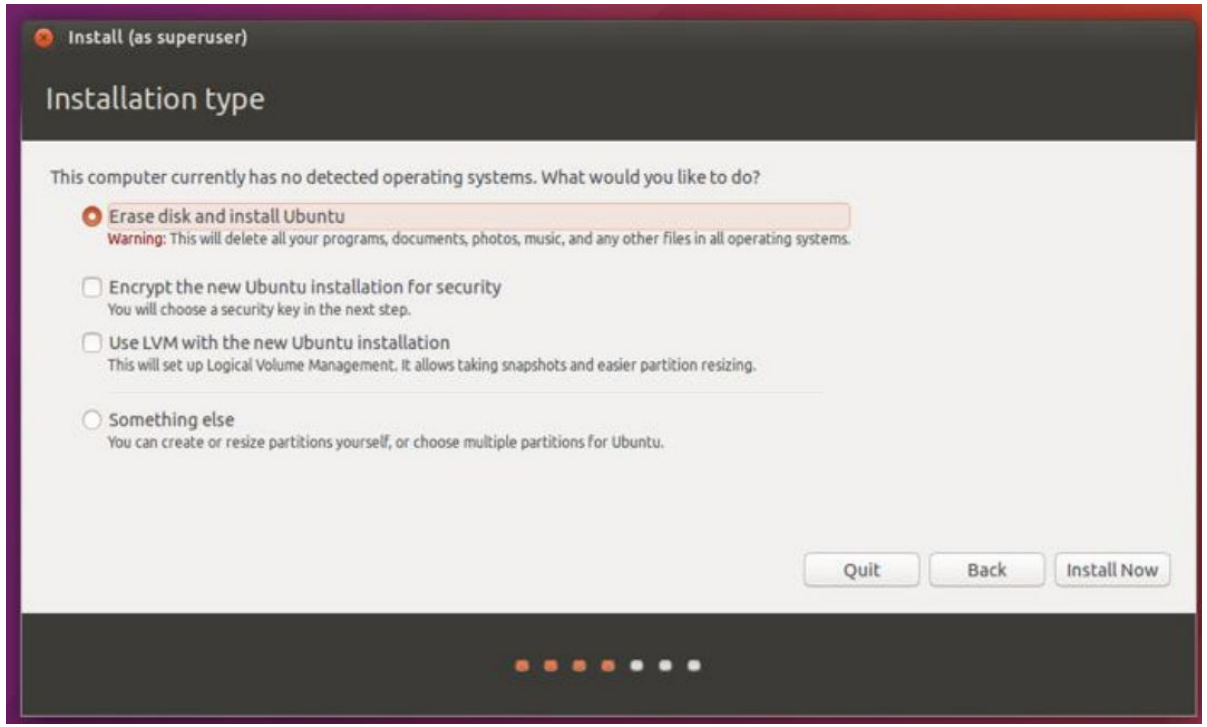


Installation

- Check Download update while installing Ubuntu.
- You may also check the other one. It doesn’t matter.
- From now, it is just installing Linux Ubuntu based on your own preferences...

Installation

- For easy setup, click on “Erase disk and install Ubuntu” (recommended)
- You may also choose “Something else” to format disk by yourself.
- Here is my setting:
  - /boot 500 MB
  - /swap 2048 MB
  - / 40 GB
  - /home 60 GB



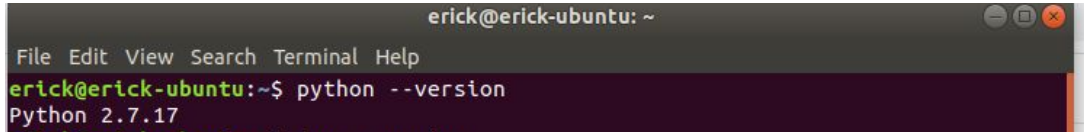
Installation

- Click on continue
- Create your account and password (**remember it for future use**)
- Installation is going to be automatically executed.
- Wait until completes!

Check python and java

- After Installation, restart the virtual machine
- Open terminal (Ctrl + Alt + T)
- Type following command to check your python and java version

```
$ python --version
$ java -version
```



THE UNIVERSITY OF TEXAS AT DALLAS

Java Installation

Installation Guide

Learning Outcomes

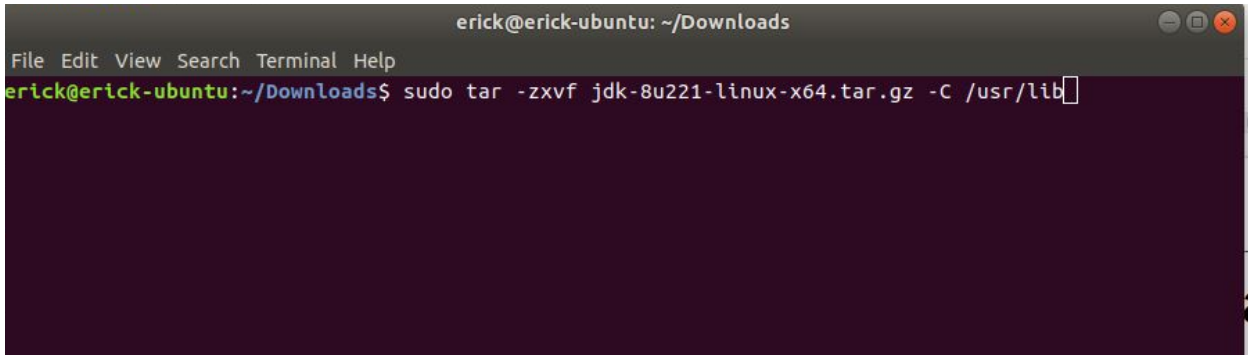
Upon completion of this lesson, students will be able to:

- Install JAVA

  Note: Hadoop is built on Java, so this step is required to be done **before** installing Hadoop.

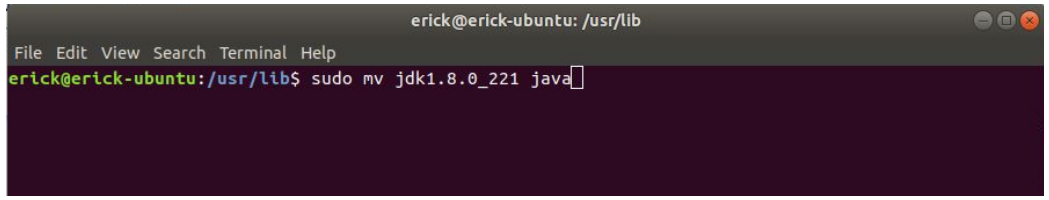
JAVA Installation

- Go to http://www.oracle.com/technetwork/java/javase/downloads/index.html to download Java SE Development Kit (my version is *jdk-8u221-linux-x64.tar.gz*).
- Remember to accept agreement and select the desired jdk file (e.g. *jdk-8u221-linux-x64.tar.gz)*
- Go to directory where you store the file and enter the command as shown below



JAVA Installation

- Rename jdk folder following commands below



- Install vim:

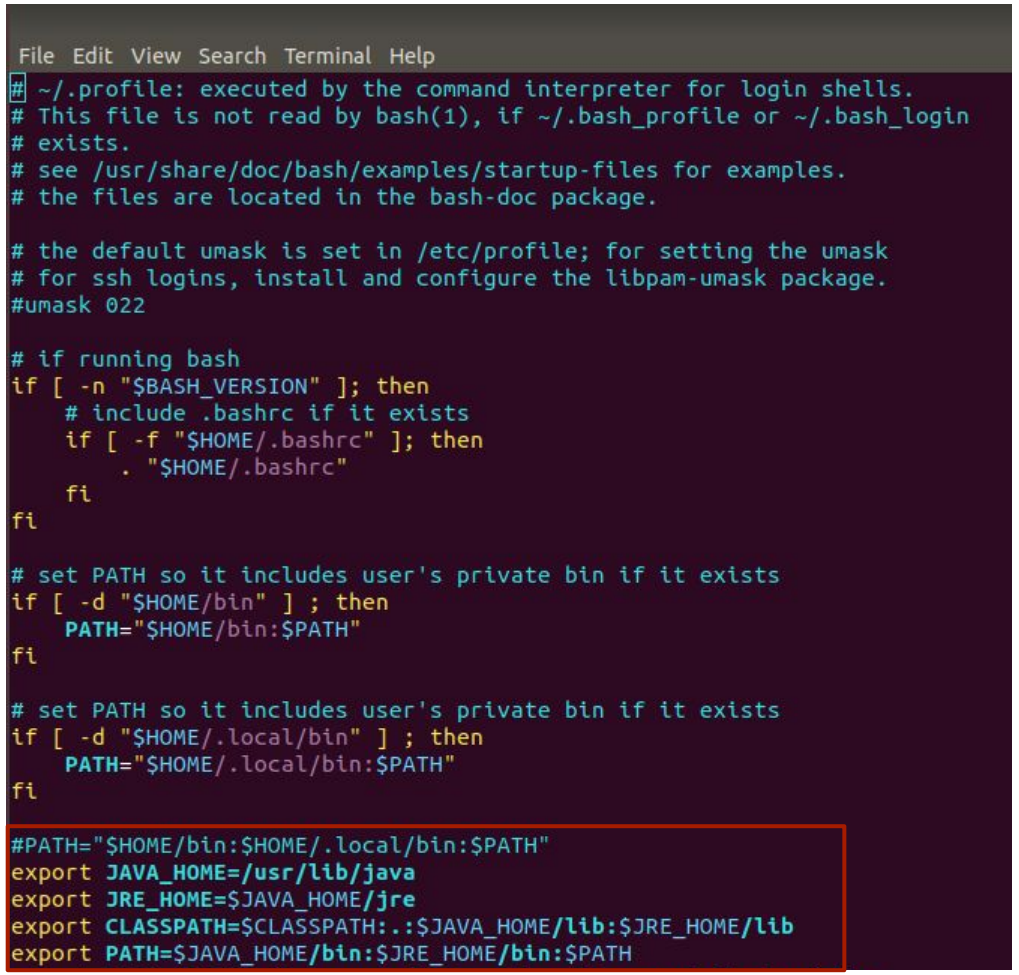
  $ sudo apt-get install vim

JAVA Installation

- Enter the following command to setup the environment variables:

  $ *vim ~/.profile*

- Enter All “**export**” commands as in the setup below at the tail of .profile
- Press ESC, then “:”
- Enter “wq”, press enter to save changes

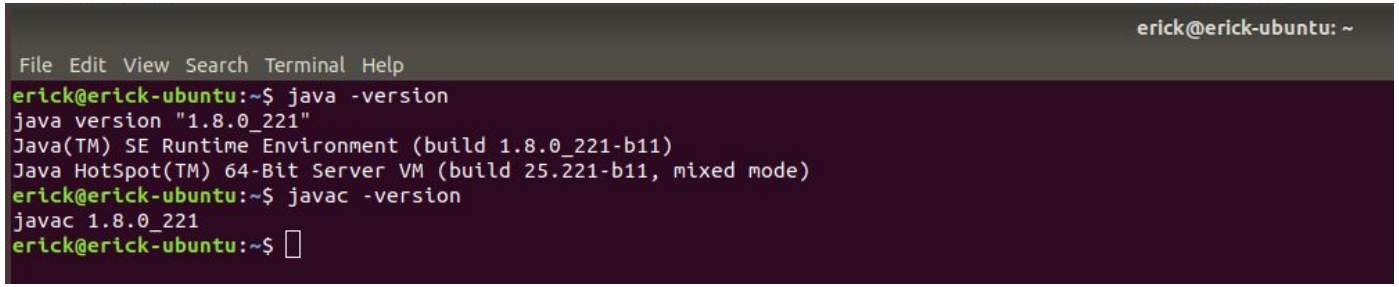


JAVA Installation

- Source *.profile*

  $ *source ~/.profile*

- Verify your installation



- Restart Ubuntu

THE UNIVERSITY OF TEXAS AT DALLAS

Hadoop Setup

Installation Guide

Learning Outcomes

Upon completion of this lesson, students will be able to:

- Install Hadoop
- Setup all configurations to run pseudo-distributed environment

Pseudo-distributed Cluster

- **Pseudo-Distributed Operation:** Hadoop can also be run on a single-node in a **pseudo-distributed mode where each Hadoop daemon runs in a separate Java process**.

- This document describes how to setup and configure a single-node Hadoop installation so that you can quickly perform simple operations using Hadoop MapReduce and the Hadoop Distributed File System (HDFS).
- Developing and testing your Hadoop program on your own machine before submitting to Hadoop cluster is **highly recommended**.
- Refer to https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html for installation documentation

Hadoop Setup

- Install **ssh** and **pdsh**:

  $ sudo apt-get install ssh
  $ sudo apt-get install pdsh

- Download Hadoop (my version is hadoop-2.8.1.tar.gz):
  https://archive.apache.org/dist/hadoop/common/hadoop-2.8.1/hadoop-2.8.1.tar.gz

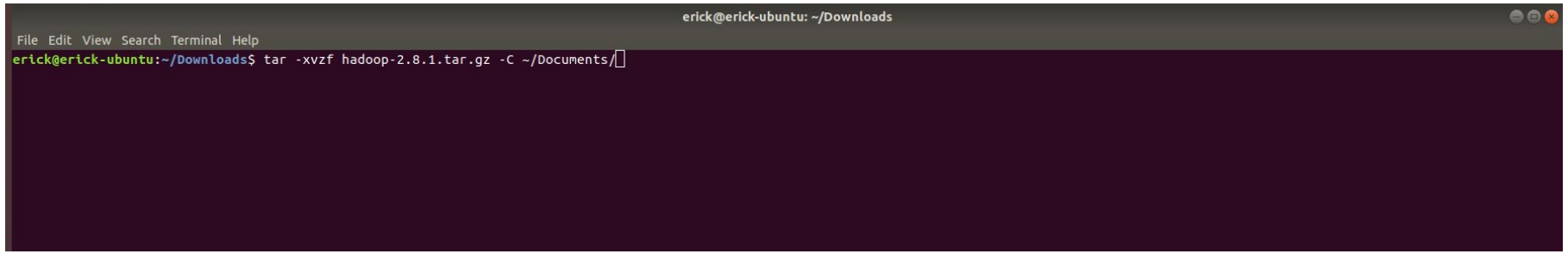
- Unzip this tar.gz file through command:

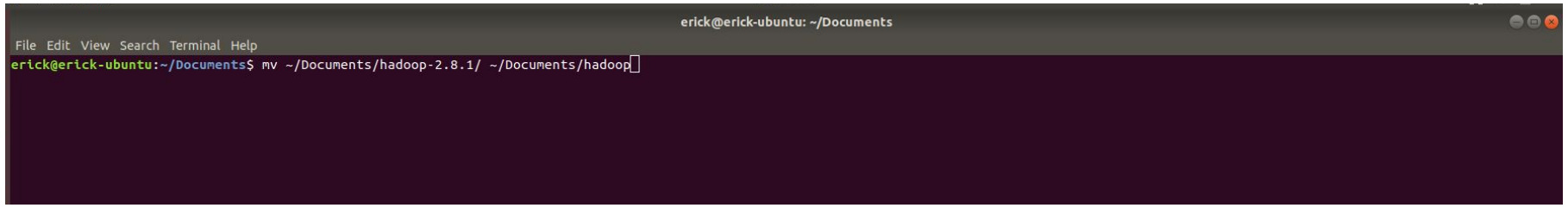
  $ tar -xvzf hadoop-2.8.1.tar.gz -C ~/Documents/

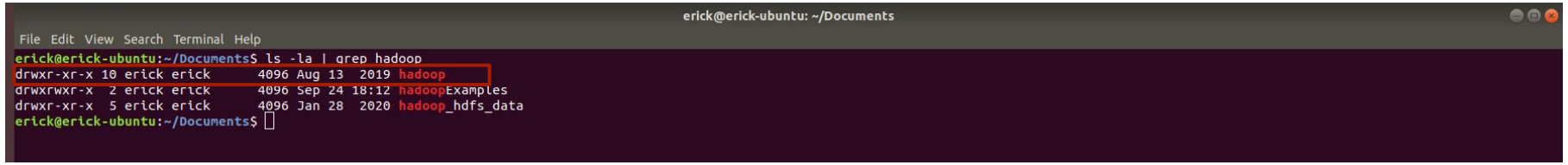
- Go to installation directory and rename the folder to hadoop

  $ mv ~/Documents/hadoop-2.8.1/ ~/Documents/hadoop

Hadoop Setup







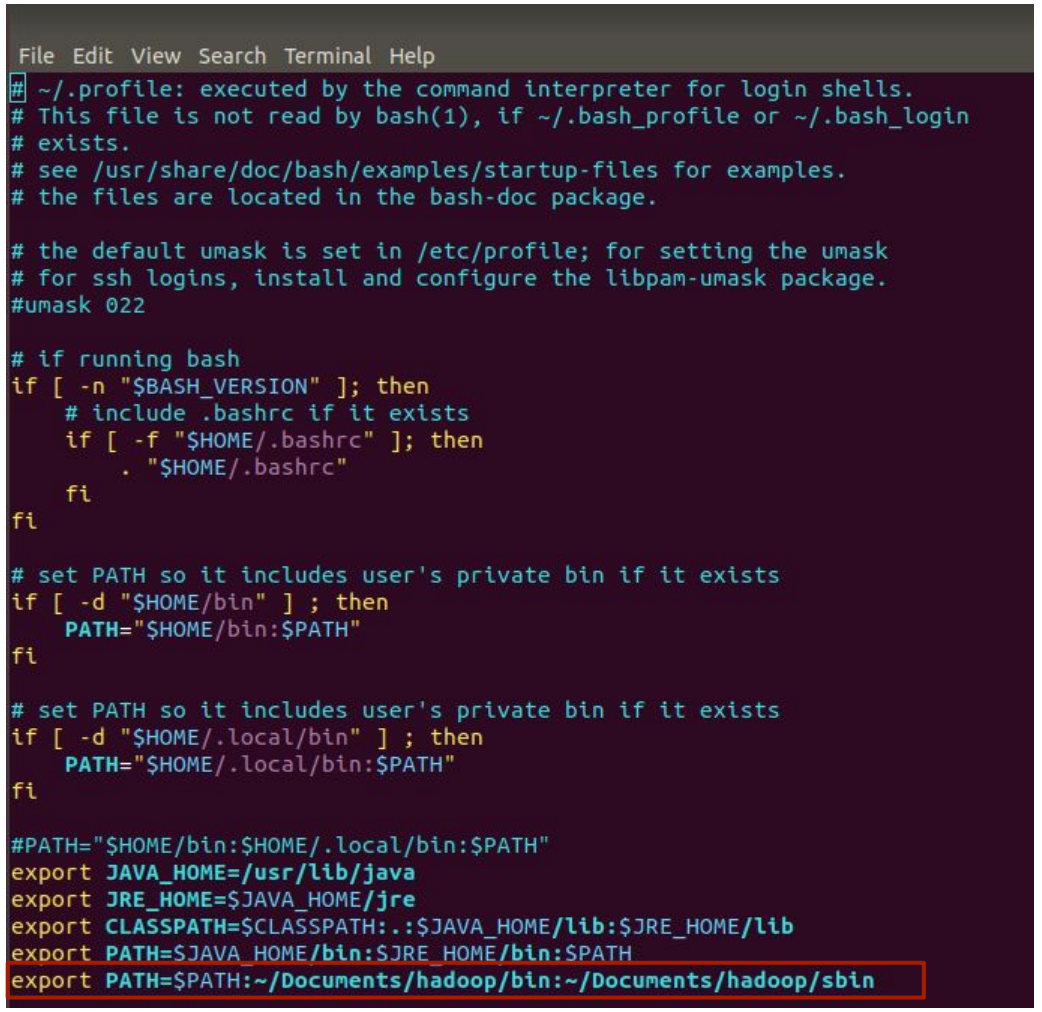
Hadoop Installation

- Enter the following command to setup the environment variables:

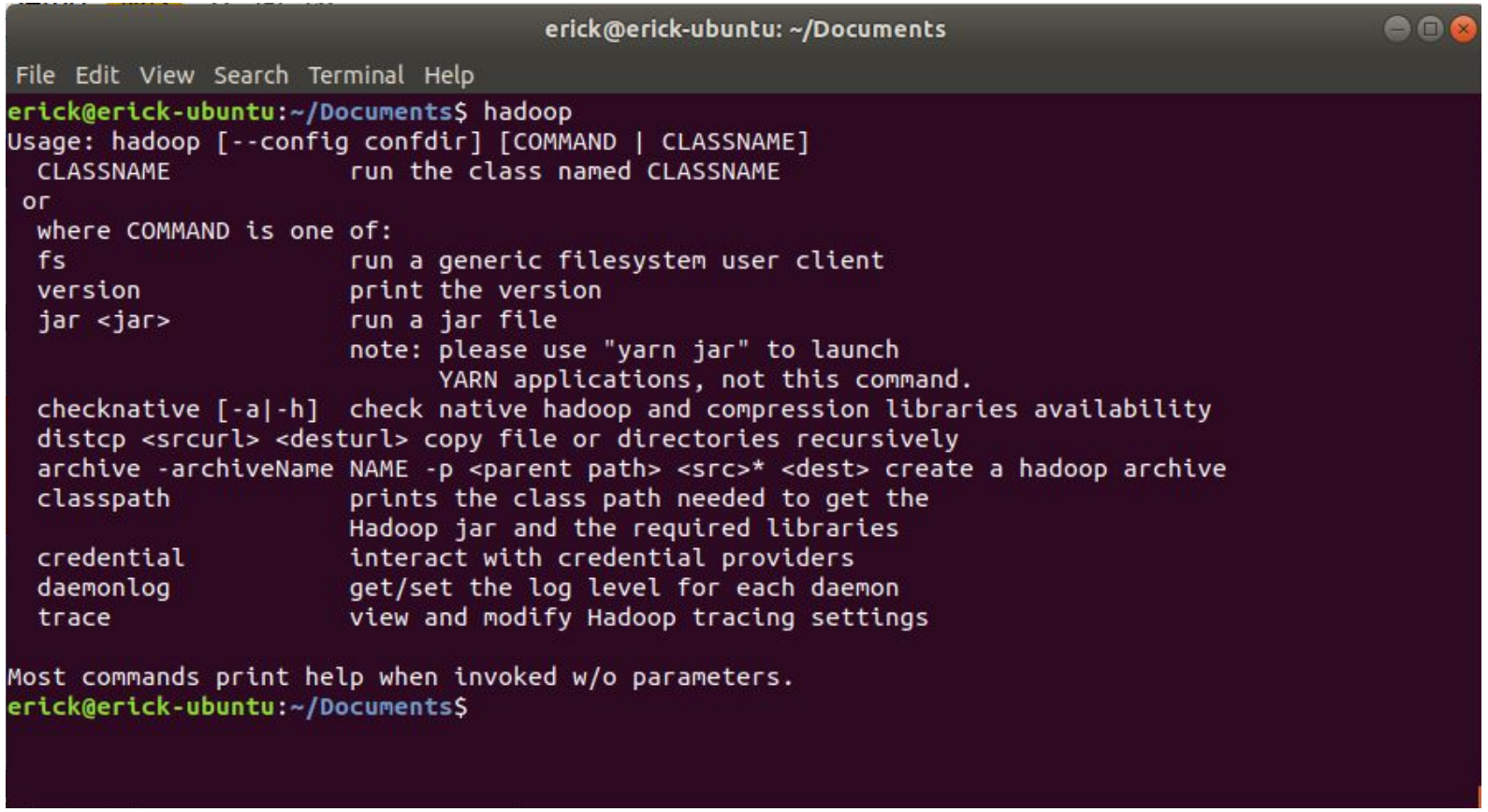
  $ *vim ~/.profile*

- Add the “export” line marked by red line at the bottom of the file
- Press ESC, then “:”
- Enter “wq”, press enter to save changes
- Source *.profile*

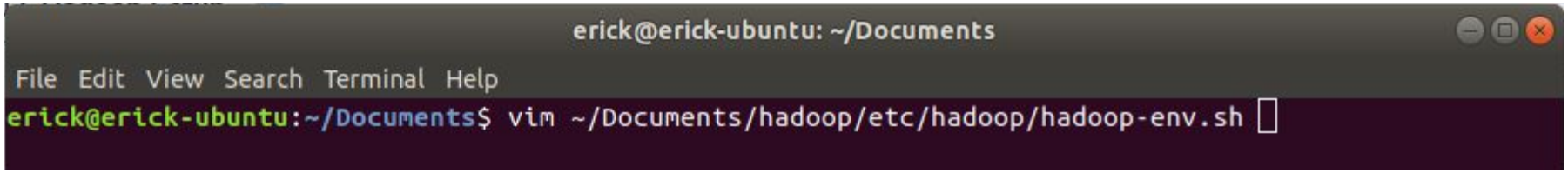
  $ *source ~/.profile*

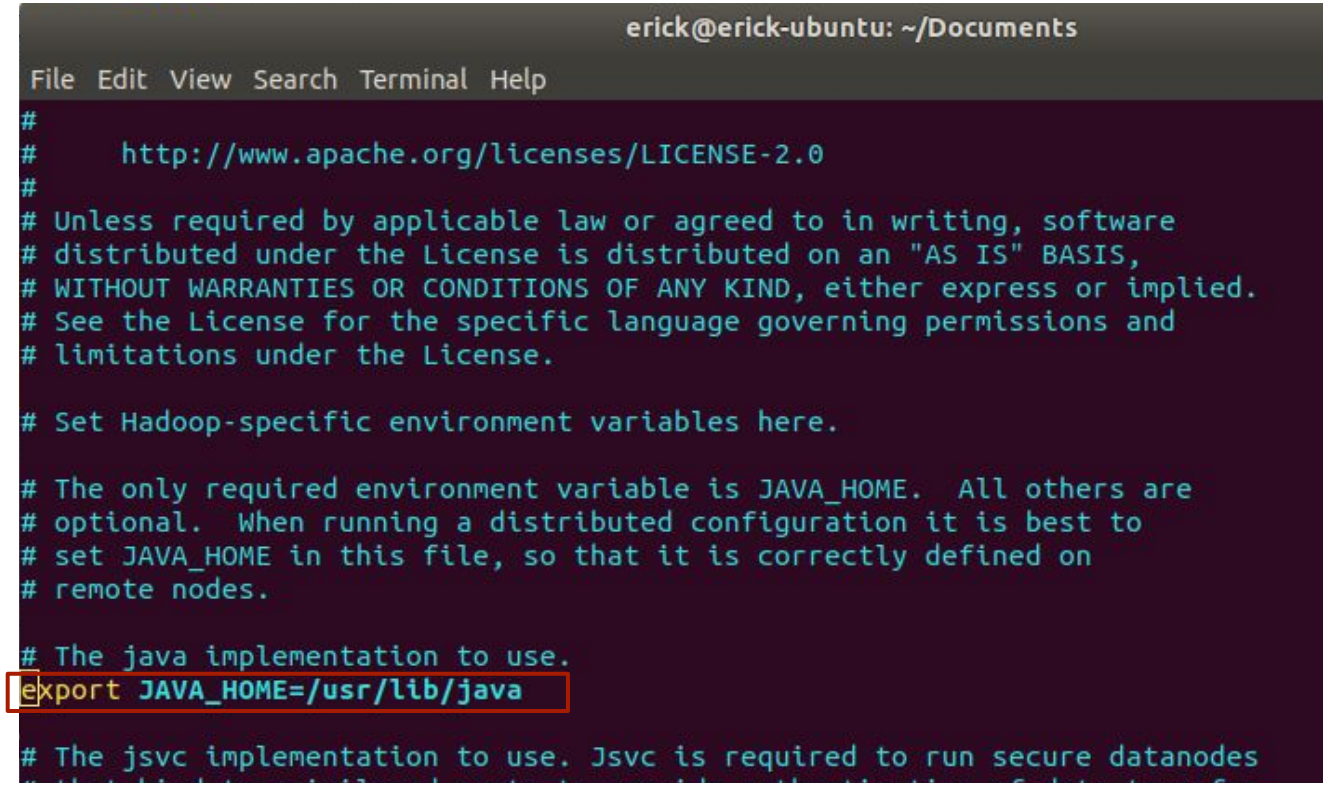


Check setup


erick@erick-ubuntu: ~/Documents
File Edit View Search Terminal Help
erick@erick-ubuntu:~/Documents$ hadoop
Usage: hadoop [--config confdir] [COMMAND | CLASSNAME]
CLASSNAME run the class named CLASSNAME
or
where COMMAND is one of:
fs run a generic filesystem user client
version print the version
jar <jar> run a jar file
note: please use "yarn jar" to launch
YARN applications, not this command.
checknative [-a|-h] check native hadoop and compression libraries availability
distcp <srcurl> <desturl> copy file or directories recursively
archive -archiveName NAME -p <parent path> <src>* <dest> create a hadoop archive
classpath prints the class path needed to get the
Hadoop jar and the required libraries
credential interact with credential providers
daemonlog get/set the log level for each daemon
trace view and modify Hadoop tracing settings
Most commands print help when invoked w/o parameters.
erick@erick-ubuntu:~/Documents$

Change JAVA_HOME in hadoop-env.sh





Same as we did before:

- Press ESC, then ":"
- Enter "wq", press enter to save changes

Pseudo-distributed mode

- Go to Hadoop folder, open **core-site.xml** and add the following configuration:

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

- Enter the content above in **core-site.xml**, save and exit (see next slide).

Pseudo-distributed mode

```
erick@erick-ubuntu: ~/Documents
File  Edit  View  Search  Terminal  Help
erick@erick-ubuntu:~/Documents$ vim ~/Documents/hadoop/etc/hadoop/core-site.xml
```

```
erick@erick-ubuntu: ~/Documents
File  Edit  View  Search  Terminal  Help
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
        <property>
                <name>fs.defaultFS</name>
                <value>hdfs://localhost:9000</value>
        </property>
</configuration>
~
~
~
~
```

Same old to save core-site.xml: press ESC, press ":", enter "wq", and press enter to save changes

Pseudo-distributed mode

- Now, we need to create a directory in the local environment for hosting datanodes and namenodes (meta)data:

  $ mkdir <directory for hosting metadata for hadoop>

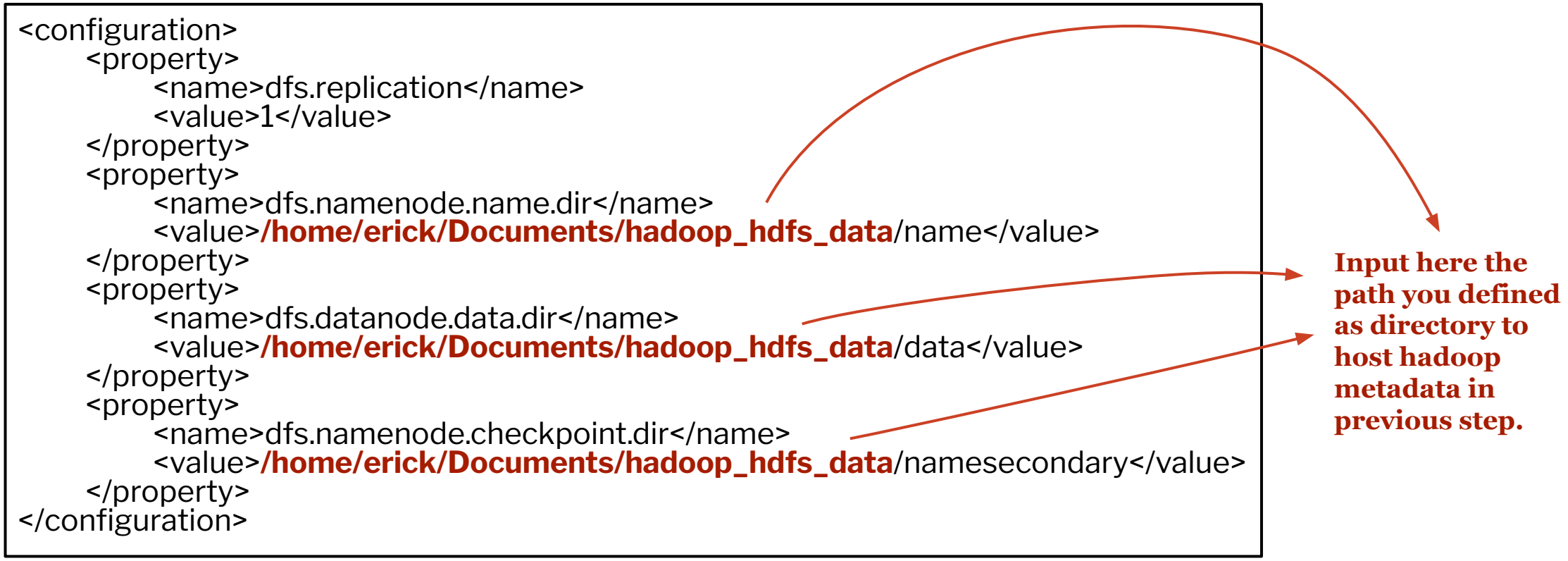
  In my case: $ mkdir **/home/erick/Documents/hadoop_hdfs_data*

- Keep this path!! We will use it in the next step...

Pseudo-distributed mode

- Go to Hadoop folder, open **hdfs-site.xml** and add the following configuration:

```
<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>
        <property>
                <name>dfs.namenode.name.dir</name>
                <value>/home/erick/Documents/hadoop_hdfs_data/name</value>
        </property>
        <property>
                <name>dfs.datanode.data.dir</name>
                <value>/home/erick/Documents/hadoop_hdfs_data/data</value>
        </property>
        <property>
                <name>dfs.namenode.checkpoint.dir</name>
                <value>/home/erick/Documents/hadoop_hdfs_data/namesecondary</value>
        </property>
</configuration>
```



Pseudo-distributed mode

```
erick@erick-ubuntu: ~/Documents
File  Edit  View  Search  Terminal  Help
erick@erick-ubuntu:~/Documents$ vim ~/Documents/hadoop/etc/hadoop/hdfs-site.xml
```

```
erick@erick-ubuntu: ~/Documents
File  Edit  View  Search  Terminal  Help
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>

        <property>
            <name>dfs.namenode.name.dir</name>
            <value>/home/erick/Documents/hadoop_hdfs_data/name</value>
        </property>

        <property>
            <name>dfs.datanode.data.dir</name>
            <value>/home/erick/Documents/hadoop_hdfs_data/data</value>
        </property>

        <property>
            <name>dfs.namenode.checkpoint.dir</name>
            <value>/home/erick/Documents/hadoop_hdfs_data/namesecondary</value>
        </property>

</configuration>
~
~
```

Same old to save hdfs-site.xml:
press ESC, press ":", enter "wq", and press enter to save changes

Pseudo-distributed mode

- ssh to localhost with the following command:

  $ ssh localhost

- Format the namenode with the following command:

  $ hdfs namenode -format

- Start hadoop *deamons* with the following command(s):

  $ start-all.sh

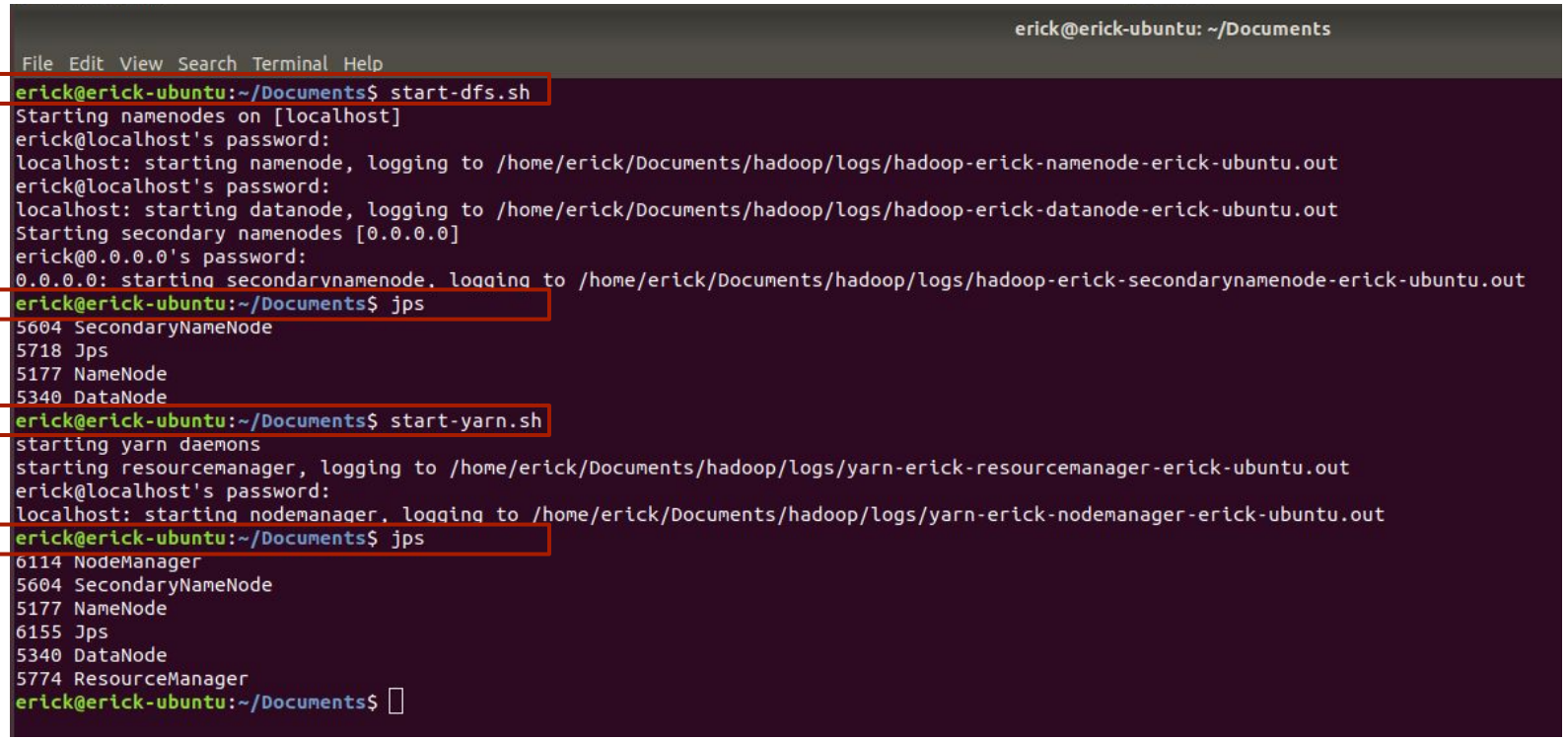
  or

  $ start-dfs.sh
  $ start-yarn.sh

Pseudo-distributed mode

start-dfs.sh command will start NameNode and DataNode deamons.
Note: use command jps to check which deamons are running.

start-yarn.sh command will start NodeManager and ResourceManager deamons.
Again: use command jps to check which deamons are running.

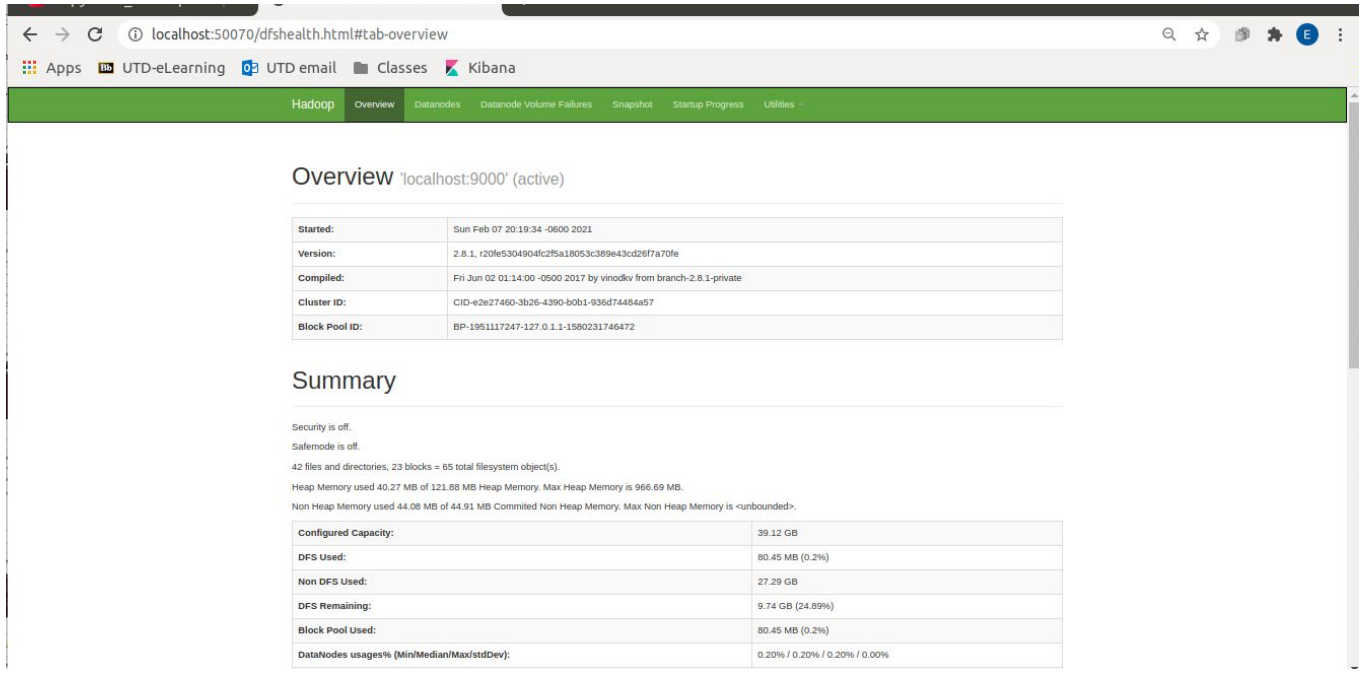


Make sure all deamons NameNode, DataNode, NodeManager and ResourceManager are running.

Pseudo-distributed mode

- Browse the web interface for the NameNode:
  - by default it is available at: ***http://localhost:50070***

    (or *http://localhost:9870* in the case you are using *Hadoop 3.0.0 or above*).



Pseudo-distributed mode

- To stop your hadoop system, you can execute the following command(s):

```
$ stop-all.sh

      or
$ stop-dfs.sh
$ stop-yarn.sh
```

THE UNIVERSITY OF TEXAS AT DALLAS

WordCount Example

Designing our first Hadoop code

Learning Outcomes

Upon completion of this lesson, students will be able to:

- Design and code our first Hadoop job

Pseudo-code

```
map(String input_key, String input_value):
    // input_key: document name
    // input_value: document contents
    for each word w in input_value:
        emit(w, "1");

reduce(String output_key, Iterator intermediate_values):
    // output_key: a word
    // output_values: a list of counts
    key = output_key
    int result = 0;
    for each v in intermediate_values:
        result = result + ParseInt(v);
        emit(key, String(result));
```

Map code in WordCount.java

```
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);
        }
    }
}
```

- **map class** implements a public map method, that processes one line at a time from input data;
- Splits each line into **tokens separated by whitespaces**.
- It **emits a key-value pair** of < <Word>, 1>, written to the Context.
- **Note**: Context object allows the Mapper to interact with the rest of the Hadoop system.

Reduce code in WordCount.java

```
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);
   }
}
```

-The **intermediate key-value pair gets grouped by based on the key** and **each group comes at a time to a reducer**

- Reducer.reduce( ) is called once per key

- Reducer.reduce( ) receives an **Iterable** which returns all values associated with that key and here we sum all those values.

- Finally, emits output with through Context.write() with all the results summarized/aggregated for each key.

Driver setup in WordCount.java

```
//   Usage: wordcount <in> <out>
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);


    job.setMapperClass(TokenizerMapper.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);
}
```

THE UNIVERSITY OF TEXAS AT DALLAS

Running Hadoop Program

Running WordCount Job

Learning Outcomes

Upon completion of this lesson, students will be able to:

- Install and setup IDE
  - Eclipse
  - Intellij
- Create a Hadoop program
- Run a simple MapReduce Program

Introduction

- So far from previous tutorials, we have a pseudo-distributed cluster running on our machine and learned how to write our first map-reduce program
- Next Questions is
  - How to get the jar of the map-reduce program?
  - How to run the jar file in Hadoop?

The Process

- We assume that you already have Hadoop on your own machine.
- In the following, we will discuss steps in details

Preparing IDE

- Hadoop programs are Java programs. You may use any Java IDE such as Eclipse, NetBeans, IntelliJ IDEA to develop your Map-Reduce program.

- We are going to use Eclipse/Intellij in this tutorial.

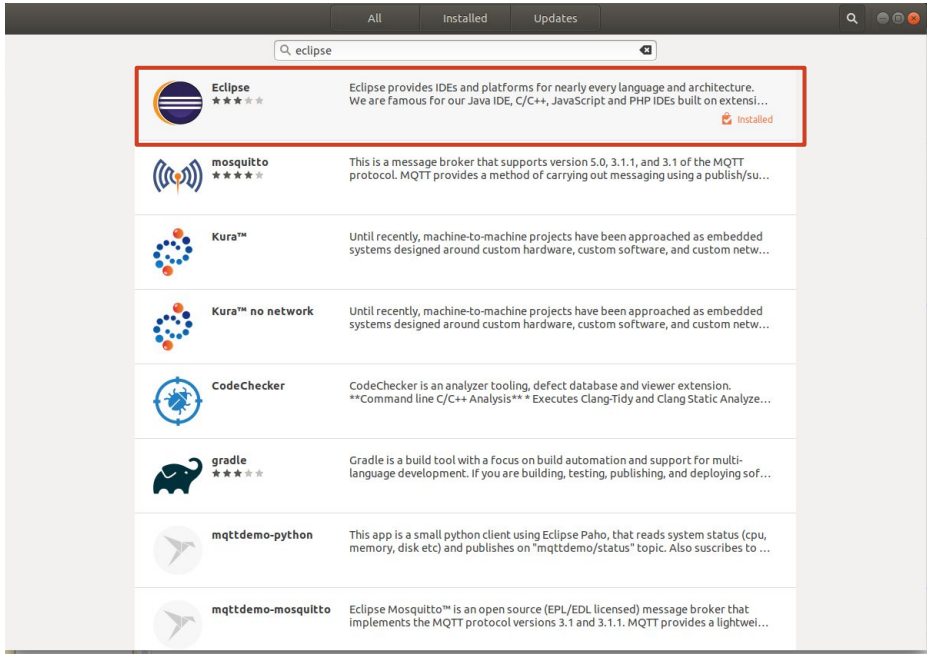
- If you have Eclipse/Intellij on your own machine, you can skip this section.

Install Eclipse

- To install Eclipse, run this command in shell

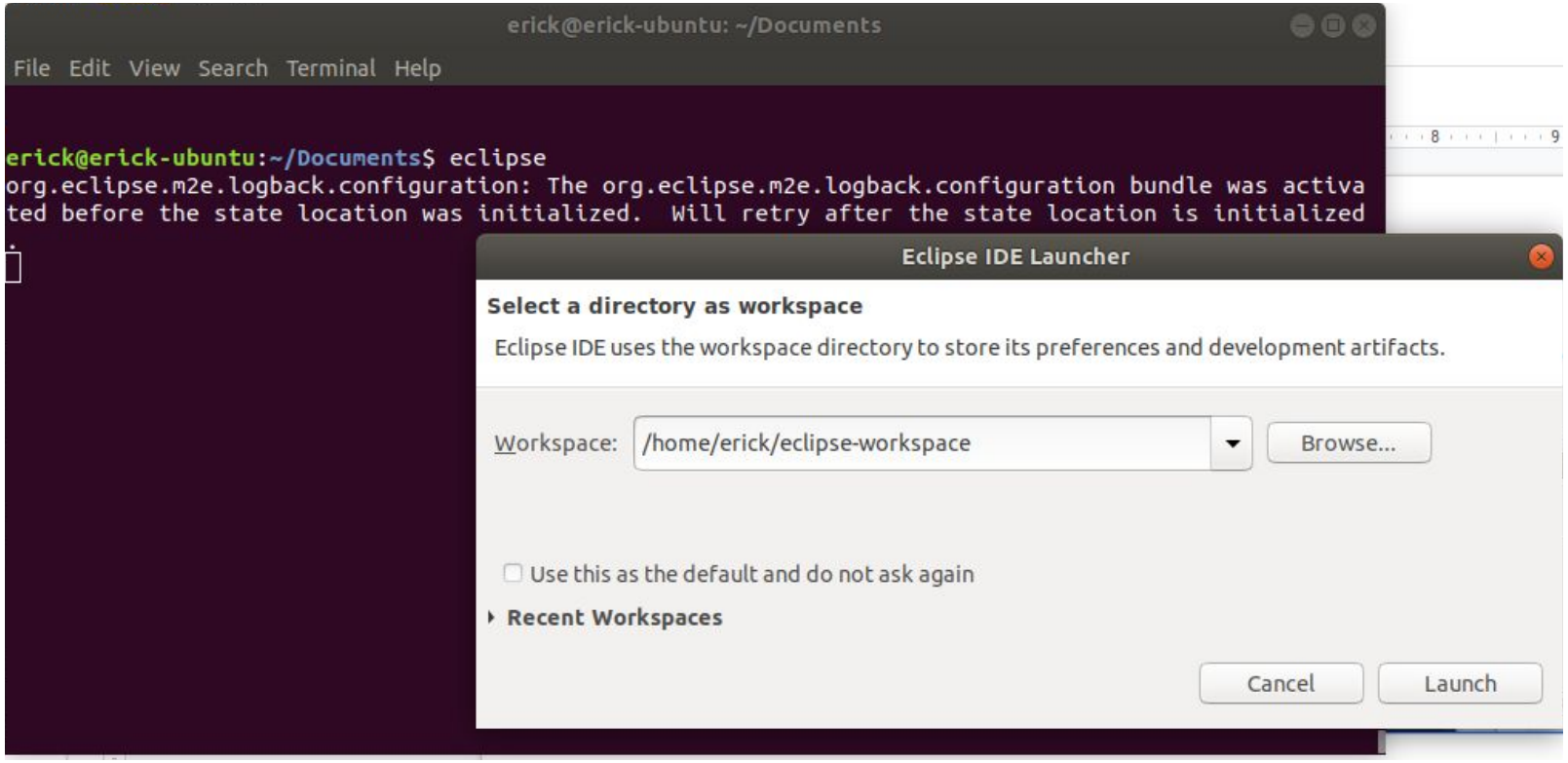
  $ sudo apt-get install eclipse

... or simply use the Ubuntu Software  for that:



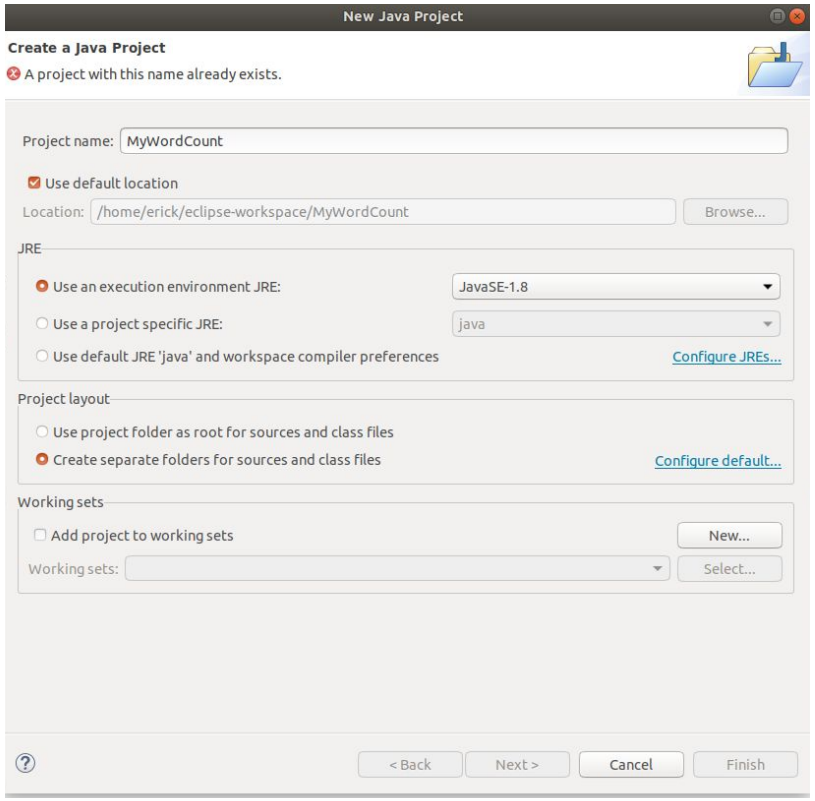
Start Eclipse

Run command “$ eclipse” to start Eclipse:



Create Project in Eclipse

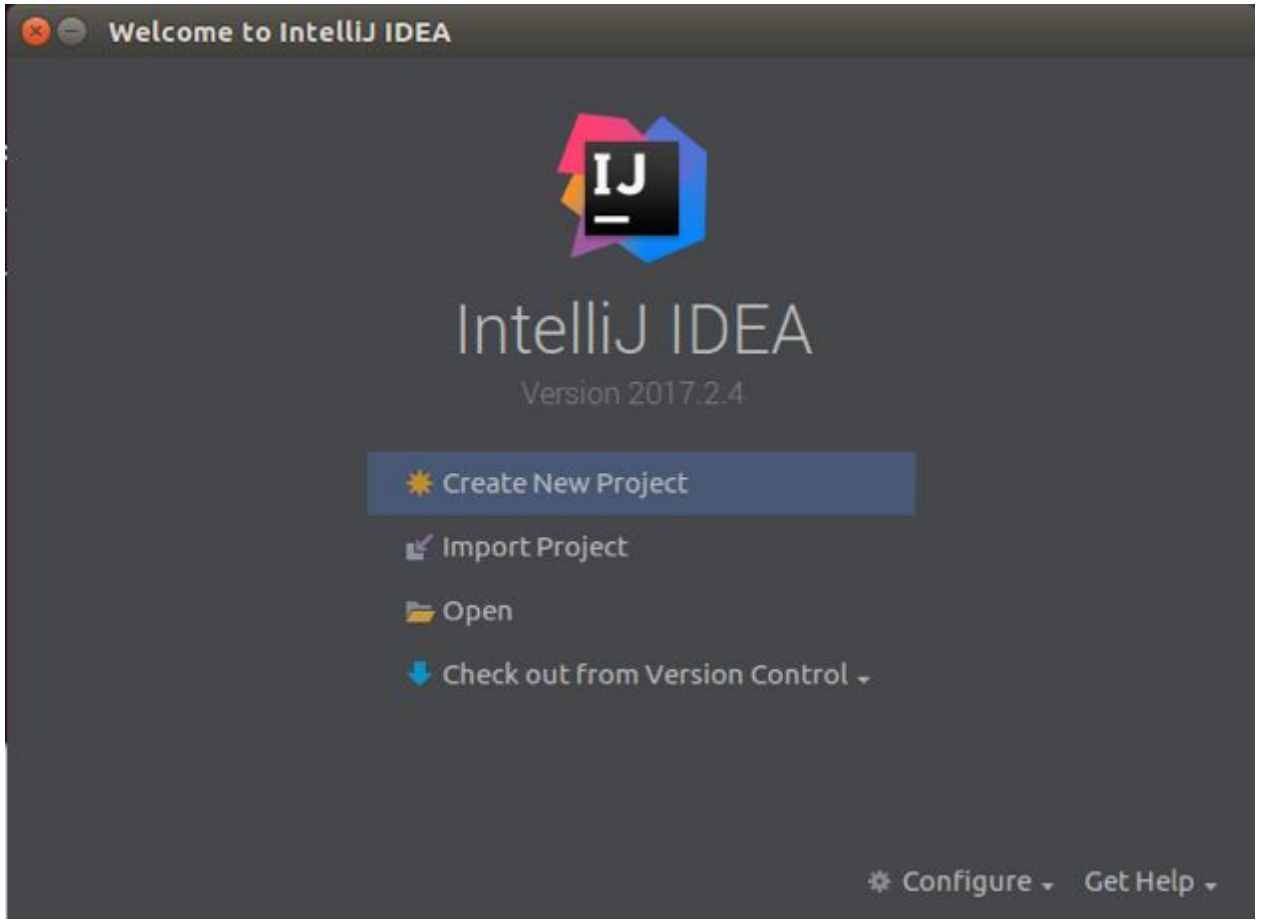
File >> New >> Java Project to create your new project...



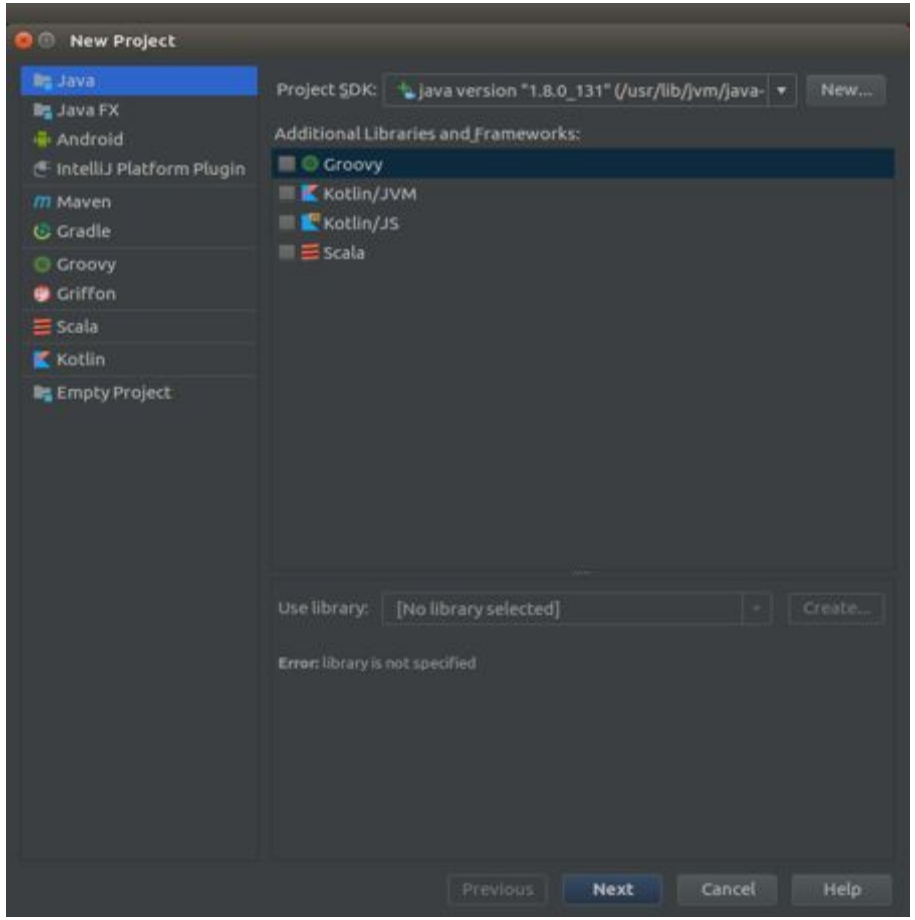
Install Intellij

- Download intellij community version
  https://www.jetbrains.com/idea/download/#section=linux
- Double click on the downloaded tar.gz file, unzip it to a folder
- Go to that folder then go to the bin folder (e.g. idea-IC-172.4155.36/bin)
- run the command “./idea.sh” in shell
- Remember to install scala plugin if you use Intellij
- For Intellij, you can also use Ubuntu Software 

Start Intellij



Create Project in Intellij



Click on next until done...

Create your JAVA Class

- Create a new java class
- Copy all the code in **WordCount.java** into this file

Create your JAVA Class

- Create a new java class
- Copy all the code in the attached **WordCount.java** into this new class file

```
public static class Map
        extends Mapper<LongWritable, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text(); // type of output key

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] mydata = value.toString().split(" ");
        for (String data : mydata) {
            word.set(data); // set word as each input keyword
            context.write(word, one); // create a pair <keyword, 1>
        }
    }
}
```

Create your JAVA Class

```java
public static class Reduce
        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; // initialize the sum for each keyword
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result); // create a pair <keyword, number of occurences>
    }
}
```

```java
// Driver program
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    // get all args
    if (otherArgs.length != 2) {
        System.err.println("Usage: WordCount <in> <out>");
        System.exit(2);
    }

    // create a job with name "wordcount"
    Job job = new Job(conf, "wordcount");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    // uncomment the following line to add the Combiner job.setCombinerClass(Reduce.class);

    // set output key type
    job.setOutputKeyClass(Text.class);
    // set output value type
    job.setOutputValueClass(IntWritable.class);
    //set the HDFS path of the input data
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    // set the HDFS path for the output
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    //Wait till job completion
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

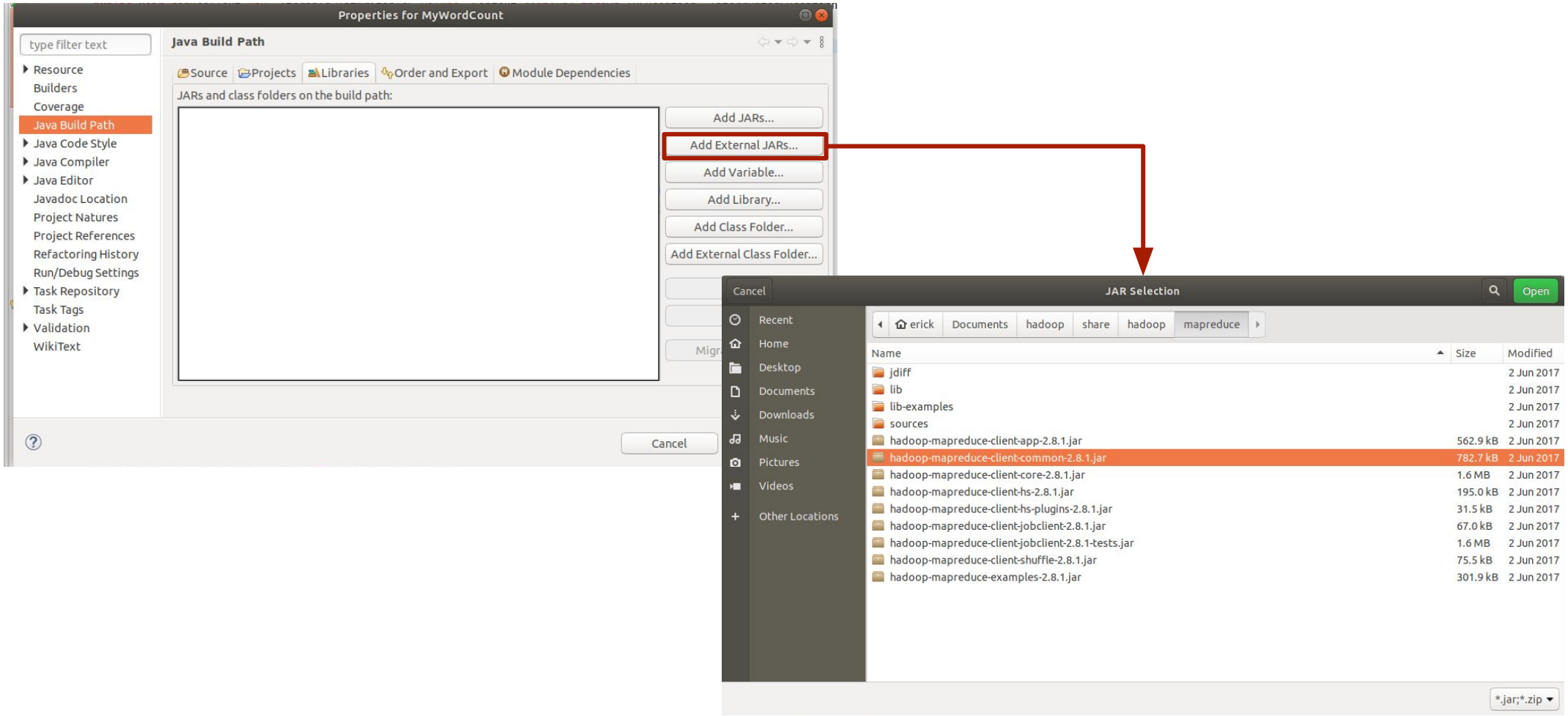
Adding Hadoop libraries

In order to compile Hadoop projects, you need to add Hadoop libraries as a reference to your projects...

Adding Hadoop libraries in Eclipse

- Right-click on the project:
  - Select "**Build Path**" >> "**Configure Build Path...**"
  - Select "**Libraries**" tab.
  - Click on "**Add External JARs...**" to continue.
- Go to your Hadoop installation path (my case is ~/Documents).
- Find "**hadoop-mapreduce-client-core-2.8.1.jar**" in */share/hadoop/mapreduce* folder, and add it.
- Click on "Add External JARs..."again.
- Find "**hadoop-common-2.8.1.jar**" in */share/hadoop/common* folder, and add it.
- You need also add "**commons-cli-1.2.jar**" in the folder */share/hadoop/common/lib* folder.

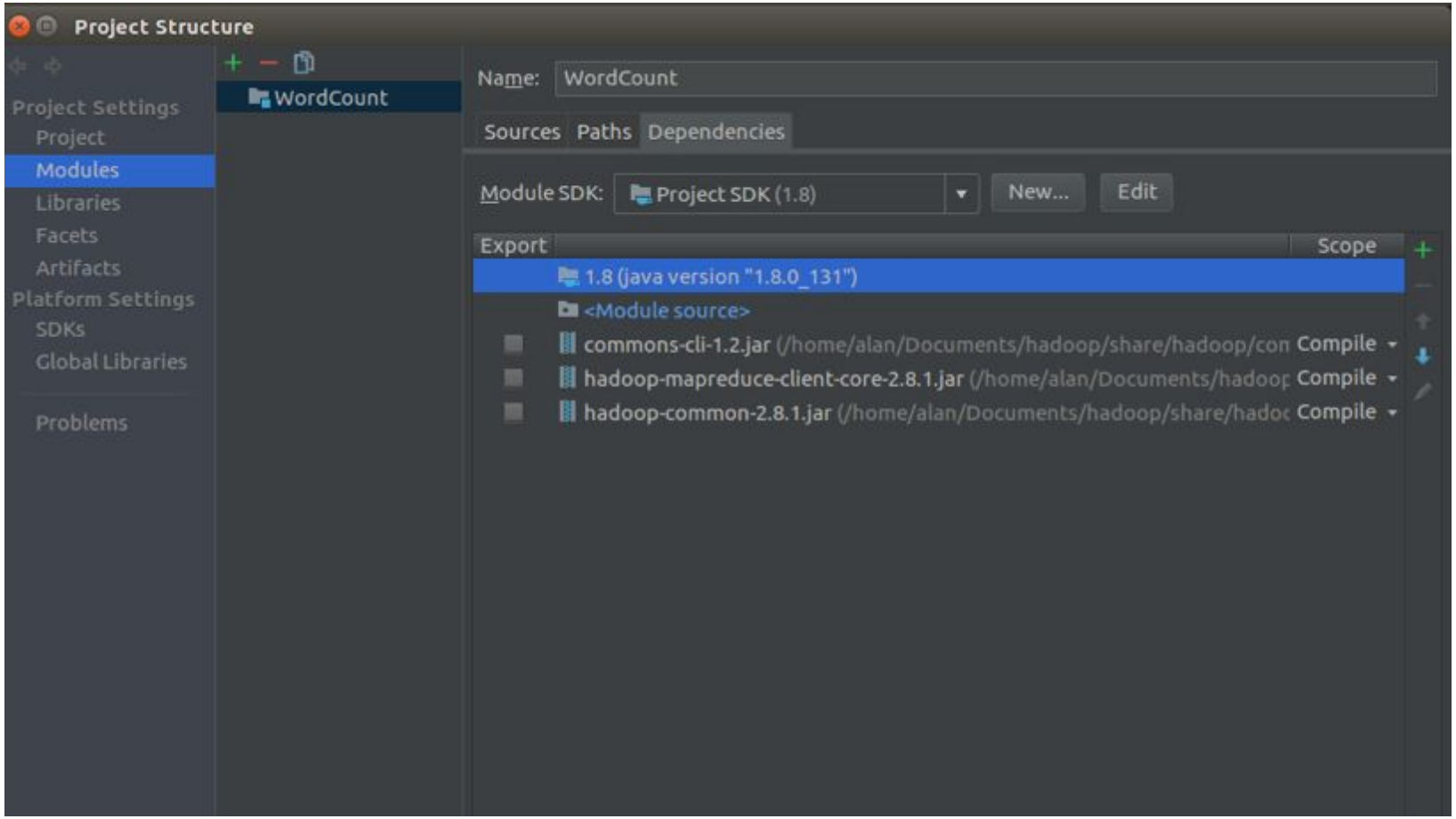
Adding Hadoop libraries in Eclipse



Adding Hadoop libraries in Intellij

- Click on File -> Project Structure -> Module
- On the larger right pane, choose the Dependencies tab
- Click on “+” and choose ”Jars or directories”
- Add all those jars mentioned above
- Click on OK to save and exit

Adding Hadoop libraries in Intellij



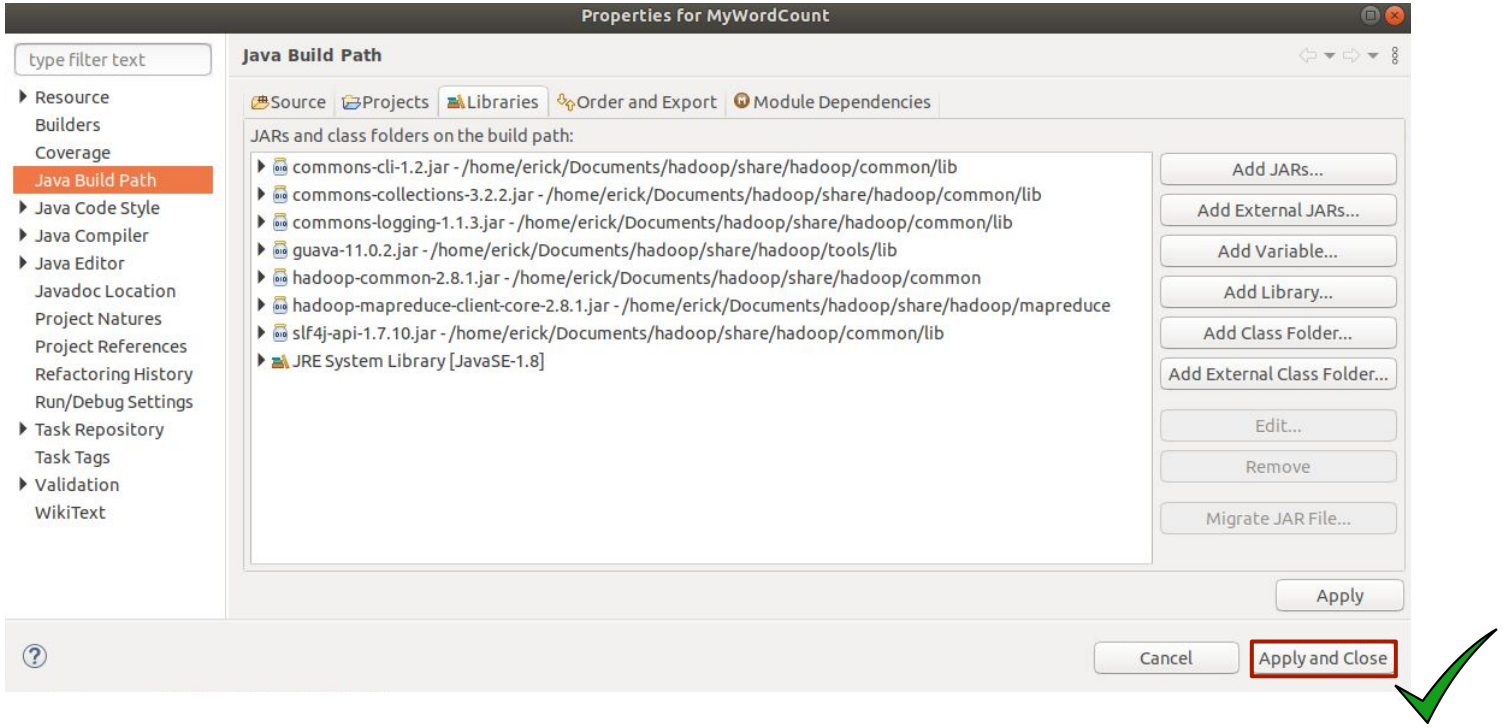
Additional Hadoop libraries in both (Eclipse and Intellij)

- Add all the JARS below into build path (both Eclipse and Intellij)

  - **commons-collections-3.2.2.jar**
  - **slf4j-api-1.7.25.jar (or slf4j-api-1.7.10.jar)**
  - **commons-logging-1.1.3.jar** at */share/hadoop/common/lib*
  - **guava-11.0.2.jar** at */share/hadoop/tools/lib*

Additional Hadoop libraries in both (Eclipse and Intellij)

- Your final build path configuration should be similar to this now:



- Note: the version of the libraries may vary depending on your hadoop version

Creating the JAR file for Hadoop

- Now, all you need to do now is to create the JAR file and run it in Hadoop...

Creating the JAR file for Hadoop in Eclipse

- Right click on the project, and choose "**Export...**"
- Then use choose "**JAR file**" under "**Java**" and click **Next.**
- Specify the name of the export file:
  - For example, you may use: "~/Documents/hadoopExamples/MyWordCount.jar"
  - Click and Finish

Creating the JAR file for Hadoop in Intellij

- File -> Project Structure -> Artifacts -> “+” -> Jar -> From modules with dependencies
- Then select main class and click on OK.
- Next, go to Build >> Build Artifacts >> Build. It will be in out/artifacts folder

Data manipulation in HDFS

- Before running our JAR file in Hadoop, we need to make sure our input file is available in HDFS system

- Remember: HDFS is build on the top of Linux!
  - Therefore, the commands you run in HDFS will be similar to those that you would run in usual shell script in your Ubuntu

- Next you see some couple of commands that you will run on HDFS to handle data and the directory file structure...

Data manipulation in HDFS

- To create a directory:
  $ hdfs dfs -mkdir <directory name you want to create>

- To list/display a directory:
  $ hdfs dfs -ls <directory name you want to look inside>

- To remove a directory or a file:

  $ hdfs dfs -rm -r <directory name you want to delete/remove>
  $ hdfs dfs -rm <file name you want to delete/remove>

Data manipulation in HDFS

```
erick@erick-ubuntu:~/Documents$ hdfs dfs -mkdir /user
erick@erick-ubuntu:~/Documents$ hdfs dfs -mkdir /user/erick
erick@erick-ubuntu:~/Documents$ hdfs dfs -ls /user
Found 1 items
drwxr-xr-x   - erick supergroup          0 2021-02-08 09:11 /user/erick
erick@erick-ubuntu:~/Documents$ hdfs dfs -ls /
Found 11 items
drwxr-xr-x   - erick supergroup          0 2020-09-24 17:39 /input
drwxr-xr-x   - erick supergroup          0 2020-02-19 18:31 /input_old
drwxr-xr-x   - erick supergroup          0 2020-08-27 20:21 /outputCountWords
drwxr-xr-x   - erick supergroup          0 2020-01-29 17:51 /outputFilter
drwxr-xr-x   - erick supergroup          0 2020-01-29 18:58 /outputFilter1
drwxr-xr-x   - erick supergroup          0 2020-01-29 19:27 /outputFilterByDate
drwxr-xr-x   - erick supergroup          0 2020-09-24 18:19 /outputInMemory
drwxr-xr-x   - erick supergroup          0 2020-09-24 18:15 /outputReduceSide
drwxr-xr-x   - erick supergroup          0 2020-09-24 16:31 /outputRelativeOccurrence
drwxr-xr-x   - erick supergroup          0 2020-02-19 18:32 /outputYelp
drwxr-xr-x   - erick supergroup          0 2021-02-08 09:11 /user
erick@erick-ubuntu:~/Documents$ hdfs dfs -rm -r /user
Deleted /user
erick@erick-ubuntu:~/Documents$ hdfs dfs -ls /
Found 10 items
drwxr-xr-x   - erick supergroup          0 2020-09-24 17:39 /input
drwxr-xr-x   - erick supergroup          0 2020-02-19 18:31 /input_old
drwxr-xr-x   - erick supergroup          0 2020-08-27 20:21 /outputCountWords
drwxr-xr-x   - erick supergroup          0 2020-01-29 17:51 /outputFilter
drwxr-xr-x   - erick supergroup          0 2020-01-29 18:58 /outputFilter1
drwxr-xr-x   - erick supergroup          0 2020-01-29 19:27 /outputFilterByDate
drwxr-xr-x   - erick supergroup          0 2020-09-24 18:19 /outputInMemory
drwxr-xr-x   - erick supergroup          0 2020-09-24 18:15 /outputReduceSide
drwxr-xr-x   - erick supergroup          0 2020-09-24 16:31 /outputRelativeOccurrence
drwxr-xr-x   - erick supergroup          0 2020-02-19 18:32 /outputYelp
erick@erick-ubuntu:~/Documents$
```

Creating directory ***/user***

Creating directory ***/user/erick***

Listing all contents of directory ***/user***

Listing all contents of directory / (root directory)
Note that I already had many other directories in root besides */user*

Removing recursively the directory ***/user***

Listing again all contents of directory / (root directory)
Note that directory */user* is not there anymore

Data manipulation in HDFS

- Transferring/copying data files from local file system to HDFS:
  $ hdfs dfs -put <filename in local file system/source> <target directory in HDFS>

- Transferring/copying data files from HDFS to local file system:
  $ hdfs dfs -get <source file in HDFS> <target directory in local file system>

Note 1: these are only some few commands that you can run over HDFS. Eventually you may need to do other operations.

Note 2: Remember that although you see it seems to be a local system, HDFS is a distributed file system. This is really interesting!!!

Data manipulation in HDFS

```
erick@erick-ubuntu:~/Documents/hadoopExamples$ hdfs dfs -mkdir /user
erick@erick-ubuntu:~/Documents/hadoopExamples$ hdfs dfs -mkdir /user/erick
erick@erick-ubuntu:~/Documents/hadoopExamples$ hdfs dfs -mkdir /user/erick/input
erick@erick-ubuntu:~/Documents/hadoopExamples$ ls -la
total 46488
drwxrwxr-x  2 erick erick     4096 Feb  8 09:36 .
drwxr-xr-x 14 erick erick     4096 Aug 25 14:00 ..
-rw-rw-r--  1 erick erick  6488666 Aug 18  2019 big.txt
-rw-rw-r--  1 erick erick  3027486 Feb 19  2020 business.csv
-rw-rw-r--  1 erick erick     1150 Feb 12  2020 cust_details.csv
-rw-rw-r--  1 erick erick     5272 Feb 19  2020 InMemoryJoin.jar
-rw-rw-r--  1 erick erick    79205 Jan 24  2020 inputWeather.txt
-rw-r--r--  1 erick erick     4198 Feb  8 08:52 MyWordCount.jar
-rw-rw-r--  1 erick erick     4198 Aug 27 20:10 MyWordCount.jar_old
-rw-rw-r--  1 erick erick     4198 Aug 25 13:57 MyWordCount.jar_old2
-rw-rw-r--  1 erick erick     5348 Feb 12  2020 ReduceSideJoin.jar
-rw-rw-r--  1 erick erick     8288 Feb 12  2020 RelativeOccurrence.jar
-rw-rw-r--  1 erick erick 25085396 Feb 19  2020 review.csv
-rw-rw-r--  1 erick erick     4053 Jan 24  2020 texasWeather.txt
-rw-r--r--  1 erick erick     3467 Sep 24 12:10 textInput.txt
-rw-rw-r--  1 erick erick    40630 Feb 12  2020 transaction_checking.ods
-rw-rw-r--  1 erick erick    32707 Feb 12  2020 transaction_details.csv
-rw-rw-r--  1 erick erick 12729122 Feb 19  2020 user.csv
-rw-rw-r--  1 erick erick     4110 Jan 24  2020 Weather1.jar
-rw-rw-r--  1 erick erick    23316 Jan 27  2020 weatherAnalytics.jar
erick@erick-ubuntu:~/Documents/hadoopExamples$ hdfs dfs -put big.txt /user/erick/input
erick@erick-ubuntu:~/Documents/hadoopExamples$ hdfs dfs -ls /user/erick/input
Found 1 items
-rw-r--r--   1 erick supergroup    6488666 2021-02-08 09:39 /user/erick/input/big.txt
erick@erick-ubuntu:~/Documents/hadoopExamples$
```

Creating directory ***/user/erick/input***

big.txt is the file in my local system that I want to transfer to HDFS (into */user/erick/input*)

Copying *big.txt* from my local system to HDFS.

There it is !!!

Running MyWordCount.jar in Hadoop

- Now, lets execute the WordCount program in Hadoop...

- First, please make sure your hadoop system is running fine!
  - Remember: to start it, run the command below in shell

```
$ start-all.sh
      or
$ start-dfs.sh
$ start-yarn.sh
```

Running MyWordCount.jar in Hadoop

- Transfer the file you want to count the words as input into the HDFS (like we did two slides ago)

  $ hdfs dfs -put <filename in local file system/source> <target directory in HDFS>

  *In my case: *$ hdfs dfs -put big.txt /user/erick/input*

- Run the JAR file with following command:

  $ hadoop jar <path of the JAR file> <class name> <input path of file in HDFS> <directory you want to output>

  *In my case: hadoop jar MyWordCount.jar WordCount */user/erick/input*/big.txt /outputCountWords

Running MyWordCount.jar in Hadoop

```
erick@erick-ubuntu:~/Documents/hadoopExamples$ hadoop jar MyWordCount.jar WordCount /user/erick/input/big.txt /outputCountWords
21/02/08 10:07:26 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
21/02/08 10:07:26 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
21/02/08 10:07:27 INFO input.FileInputFormat: Total input files to process : 1
21/02/08 10:07:27 INFO mapreduce.JobSubmitter: number of splits:1
21/02/08 10:07:27 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1612623958_0001
21/02/08 10:07:28 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
21/02/08 10:07:28 INFO mapreduce.Job: Running job: job_local1612623958_0001
21/02/08 10:07:28 INFO mapred.LocalJobRunner: OutputCommitter set in config null
21/02/08 10:07:28 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
21/02/08 10:07:28 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
21/02/08 10:07:28 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
21/02/08 10:07:28 INFO mapred.LocalJobRunner: Waiting for map tasks
21/02/08 10:07:28 INFO mapred.LocalJobRunner: Starting task: attempt_local1612623958_0001_m_000000_0
21/02/08 10:07:28 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
21/02/08 10:07:28 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
```

• • •

```
21/02/08 10:07:34 INFO mapred.LocalJobRunner: reduce > reduce
21/02/08 10:07:34 INFO mapred.Task: Task 'attempt_local1612623958_0001_r_000000_0' done.
21/02/08 10:07:34 INFO mapred.LocalJobRunner: Finishing task: attempt_local1612623958_0001_r_000000_0
21/02/08 10:07:34 INFO mapred.LocalJobRunner: reduce task executor complete.
21/02/08 10:07:34 INFO mapreduce.Job:  map 100% reduce 100%
21/02/08 10:07:34 INFO mapreduce.Job: Job job_local1612623958_0001 completed successfully
21/02/08 10:07:34 INFO mapreduce.Job: Counters: 35
        File System Counters
                FILE: Number of bytes read=26959920
                FILE: Number of bytes written=41075547
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=12977332
                HDFS: Number of bytes written=937095
                HDFS: Number of read operations=13
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
        Map-Reduce Framework
                Map input records=128457
                Map output records=1164553
                Map output bytes=11146463
                Map output materialized bytes=13475575
                Input split bytes=111
                Combine input records=0
                Combine output records=0
                Reduce input groups=81410
                Reduce shuffle bytes=13475575
                Reduce input records=1164553
                Reduce output records=81410
                Spilled Records=2329106
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=57
                Total committed heap usage (bytes)=465313792
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=6488666
        File Output Format Counters
                Bytes Written=937095
erick@erick-ubuntu:~/Documents/hadoopExamples$
```

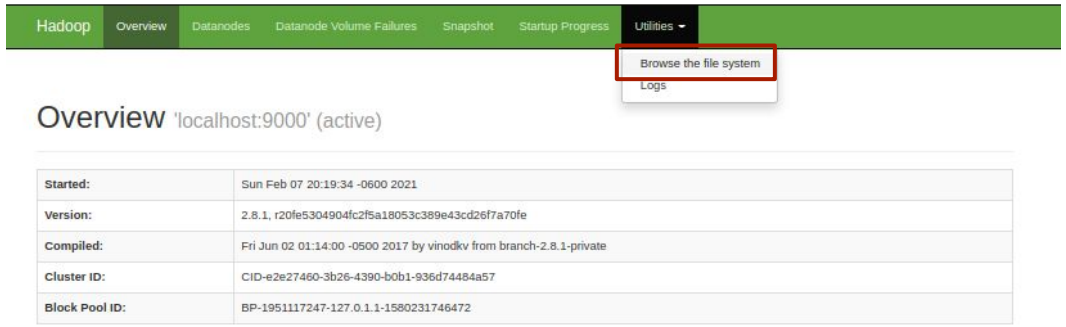
Running MyWordCount.jar in Hadoop

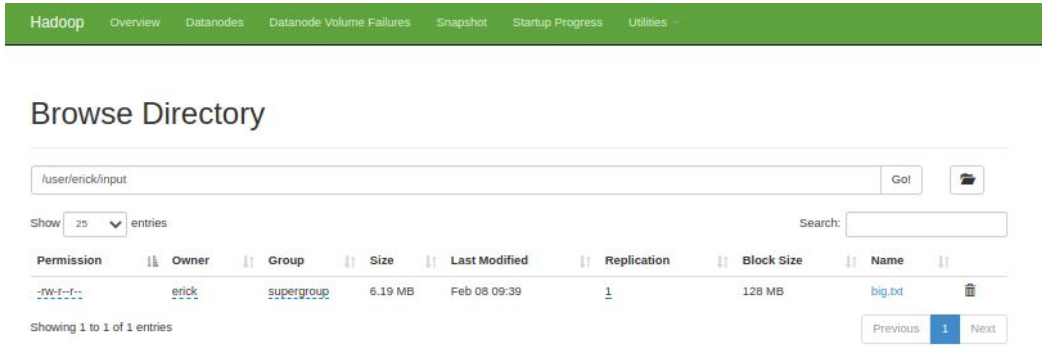
- The job was successfully run! Now we can see the output of the job and copy it to your local machine...

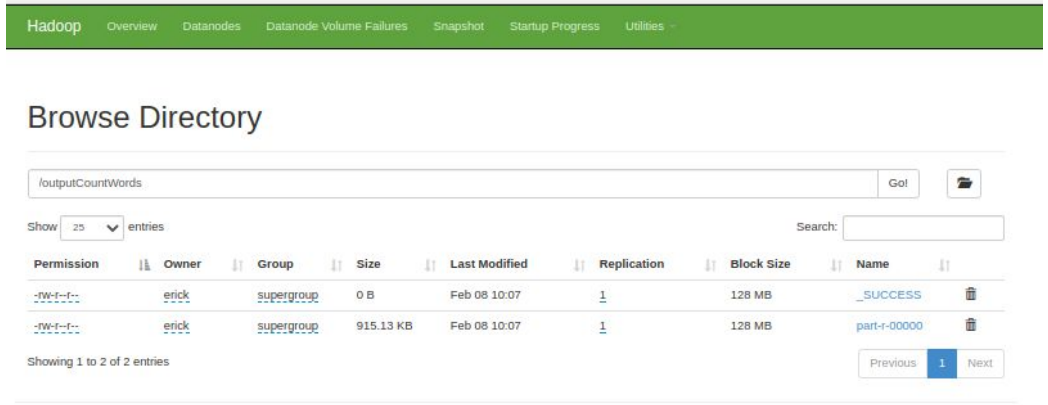
```
erick@erick-ubuntu:~/Documents/hadoopExamples$ hdfs dfs -ls /outputCountWords
Found 2 items
-rw-r--r--   1 erick supergroup          0 2021-02-08 10:07 /outputCountWords/_SUCCESS
-rw-r--r--   1 erick supergroup     937095 2021-02-08 10:07 /outputCountWords/part-r-00000
erick@erick-ubuntu:~/Documents/hadoopExamples$ hdfs dfs -get /outputCountWords/part-r-00000 /home/erick/Documents/hadoopExamples
erick@erick-ubuntu:~/Documents/hadoopExamples$ tail -25 /home/erick/Documents/hadoopExamples/part-r-00000
|Missouri       1
|Montana        1
|Nebraska       1
|Nevada 1
|New    4
|North  2
|Ohio   1
|Oklahoma       1
|Oregon 1
|Pennsylvania   1
|Rhode  1
|South  2
|Tennessee      1
|Texas  1
|United 1
|Utah   1
|Vermont        1
|Virginia       1
|Washington     1
|West   1
|Wisconsin      1
|Wyoming        1
|but    1
|had    1
|not    1
```

Running MyWordCount.jar in Hadoop

- Remember: you can always use the browser interface to navigate and check your files in HDFS:







Thank you

For questions about the material, please come to office hours or contact me:
exs172930@utdallas.edu