



CPrE/SE 419: SOFTWARE TOOLS FOR LARGE-SCALE DATA ANALYSIS, SPRING 2019

Lab 1: Using the Cluster, and Introduction to HDFS

Purpose

We have setup an environment on the “hpc-class” cluster for large-scale data analysis. The first goal of this lab is get familiar with this environment, called “Hadoop”. The second goal is to get familiar with a distributed file system for storing large data sets, called “HDFS”, which is short for “Hadoop Distributed File System”. Specifically, at the end of this lab, you will know how to:

- Login to hpc-class cluster with your user id
- Transfer files between the local machine and the cluster
- Write, compile and execute a Hadoop program
- Use HDFS and the HDFS Programming Interface

Submission

Create a zip archive, “lab1_netID.zip”, with the following and hand it in through canvas.

- A write-up answering questions for each experiment in the lab. For output, cut and paste results from your terminal and summarize when necessary.
- Commented Code for your program. Include all source files needed for compilation.

Resources

- Address of hpc-class cluster resource: hpc-class.its.iastate.edu
- Address of Hadoop Name Node: hadoop
- Hadoop library files, available on Canvas.



Logging into the Cluster (from ISU Network)

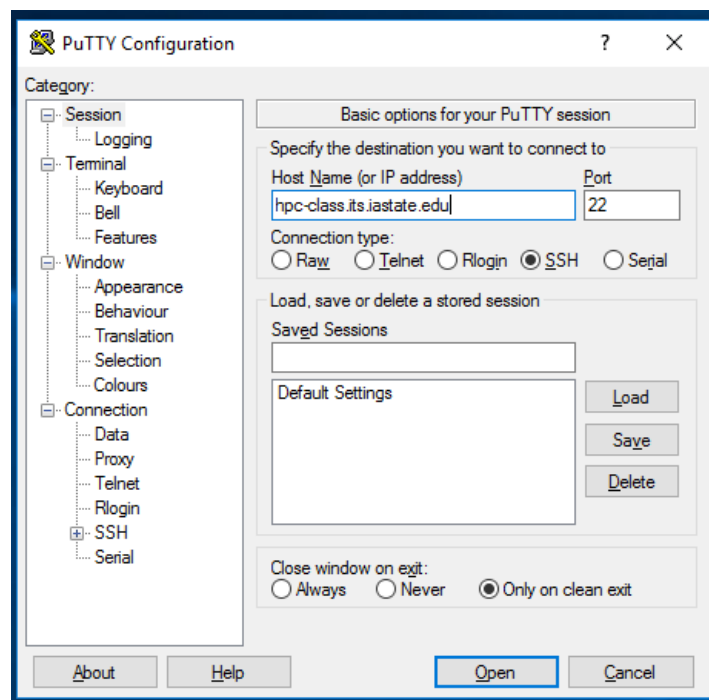
For Windows:

You can use Putty to login to the cloud. In case you do not have Putty installed in your system, you can download it from the link below:

<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

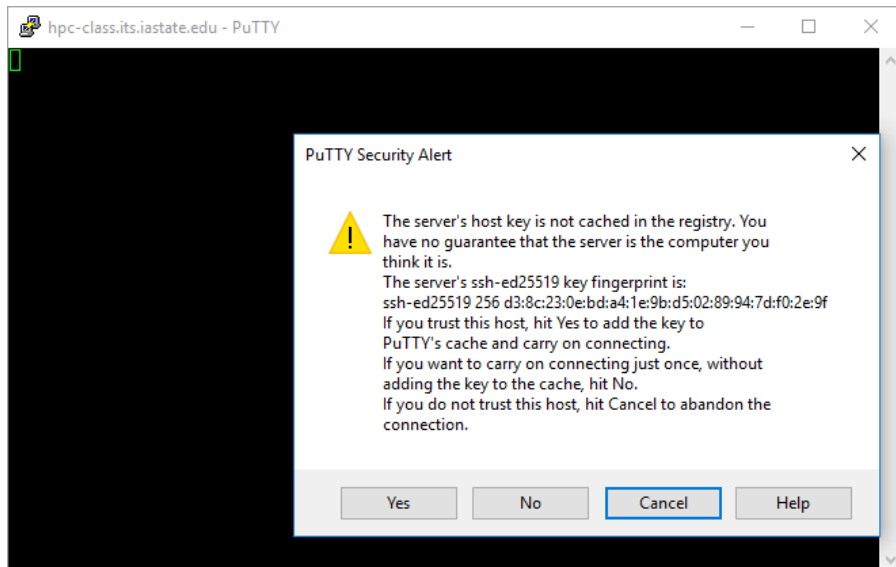
Now use Putty.exe to connect to the Cloud as follows (from ISU Network or VPN):

- Start PuTTY and give hpc-class.its.iastate.edu as the Host Name, then click Open.

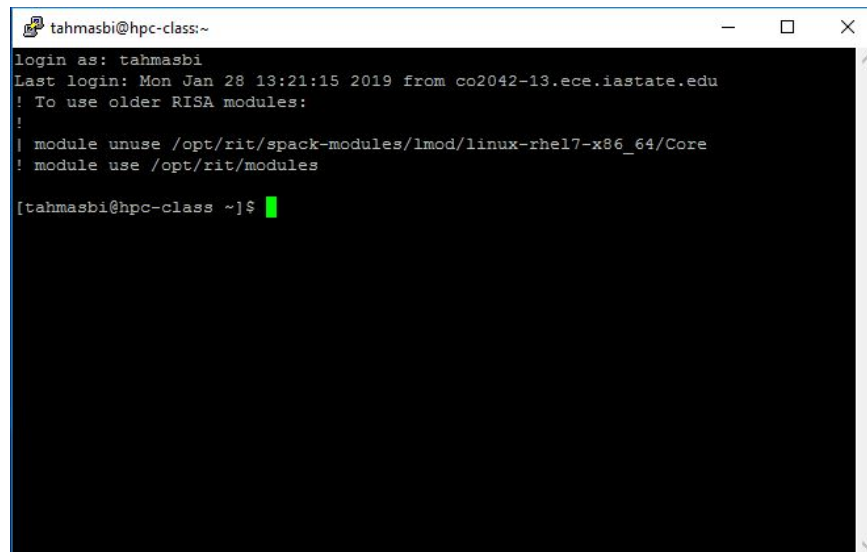




- Click "Yes" to trust the server as shown below



- Provide your ISU Net ID as the User ID and password.





- Hpc-class is a cluster of computers. We need to access the Hadoop name node on hpc-class. The next step is to ssh into namenode. Type in “ssh hadoop” to access the Hadoop name node.

```
tahmasbi@hadoop000:~  
login as: tahmasbi  
Last login: Mon Jan 28 13:21:15 2019 from co2042-13.ece.iastate.edu  
! To use older RISA modules:  
!  
! module unuse /opt/rit/spack-modules/lmod/linux-rhel7-x86_64/Core  
! module use /opt/rit/modules  
[tahmasbi@hpc-class ~]$ ssh hadoop  
Password:  
Last login: Mon Jan 28 13:21:35 2019 from hpc-class-en  
[tahmasbi@hadoop000 ~]$
```

Logging into the Cluster from outside of ISU

In case if you would like to access the cluster from outside of the campus, you need to connect to ISU network through VPN. Please follow the detailed instruction from the link below:

<https://www.it.iastate.edu/services/vpn>



HDFS Usage

HDFS is a distributed file system that is used by Hadoop to store large files. It automatically splits a file into many “blocks” of no more than 64MB and stores the blocks on separate machines. This helps the system to store huge files, much larger than any single machine can store. The HDFS also creates replicas for each block (by default 3). This helps to make the system more fault-tolerant. HDFS is an integral part of the Hadoop eco-system. HDFS is an Open Source implementation of the Google File System.

First login **Hadoop** as described in the above sections.

To try out different HDFS utilities, try the commands:

```
$ hdfs dfs -help
```

You should see something like the screenshot given below:

```
tahmasbi@hadoop000:~$ hdfs dfs -help
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hadoop/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Usage: hadoop fs [generic options]
    [-appendToFile <localsrc> ... <dst>]
    [-cat [-ignoreCrc] <src> ...]
    [-checksum <src> ...]
    [-chgrp [-R] GROUP PATH...]
    [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
    [-chown [-R] [OWNER][:[GROUP]] PATH...]
    [-copyFromLocal [-f] [-p] [-l] [-d] [-t <thread count>] <localsrc> ... <dst>]
    [-copyToLocal [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
    [-count [-q] [-h] [-v] [-t <storage type>]] [-u] [-x] [-e] <path> ...]
    [-cp [-f] [-p | -p[topax]] [-d] <src> ... <dst>]
    [-createSnapshot <snapshotDir> [<snapshotName>]]
    [-deleteSnapshot <snapshotDir> <snapshotName>]
    [-df [-h] [<path> ...]]
    [-du [-s] [-h] [-v] [-x] <path> ...]
    [-expunge]
    [-find <path> ... <expression> ...]
    [-get [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
    [-getfacl [-R] <path>]
    [-getfattr [-R] {-n name | -d} [-e en] <path>]
    [-getmerge [-nl] [-skip-empty-file] <src> <localdst>]
    [-head <file>]
    [-help [cmd ...]]
    [-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [-e] [<path> ...]]
    [-mkdir [-p] <path> ...]
    [-moveFromLocal <localsrc> ... <dst>]
    [-moveToLocal <src> <localdst>]
    [-mv <src> ... <dst>]
    [-put [-f] [-p] [-l] [-d] <localsrc> ... <dst>]
    [-renameSnapshot <snapshotDir> <oldName> <newName>]
    [-rm [-f] [-r|-R] [-skipTrash] [-safely] <src> ...]
    [-rmkdir [--ignore-fail-on-non-empty] <dir> ...]
    [-setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>][--set <acl_spec> <path>]]
    [-setfattr {-n name [-v value] | -x name} <path>]
    [-setrep [-R] [-w] <rep> <path> ...]
    [-stat [format] <path> ...]
```



You will notice that the HDFS utilities are very similar to that of the default UNIX file system utilities and hence should be easy to learn.

HDFS API

HDFS can also be accessed using a Java API, which allows us to read/write into the file system. For documentation, see: <http://hadoop.apache.org/docs/r3.1.1/api/>
Look at Package org.apache.hadoop.fs

The [FSDataOutputStream](#) and [FSDataInputStream](#) can be used to read/write files in HDFS respectively. Check out their class documentations as well.

Read the source code of the program below, that would help you to better understand how the HDFS API works. Then, look at the different classes that were used and learn more about the various methods they provide. All the methods can be found in the Hadoop API documentation. Once you have browsed through the HDFS API, go onto the following experiment.

Experiment 1 (10 points):

First, we will manually copy a file into Hadoop file system. You create a directory called /user/<your login id>/lab1 under HDFS. Create a new file called "afile.txt" on your local machine with some text. Then use WinSCP (or scp on Linux) to move the file to your home directory in hpc-class (see page 9 for more details). After that, use scp (see page 9 for more details) to move the file from your home directory in hpc-class to your home directory in hadoop cluster. Finally, use *hdfs dfs* command to move this file to the directory you created (/user/<your login id>/lab1). Show the instructor that you can manually copy file into Hadoop file system.



Next, we will write a program that create a file and write something to it. We use the Java programming language, as that is the default language used with Hadoop. The following program creates a new file and write something into it.

```
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.*;

public class HDFSWrite {
    public static void main ( String [] args ) throws Exception {
        // The system configuration
        Configuration conf = new Configuration();

        // Get an instance of the Filesystem
        FileSystem fs = FileSystem.get(conf);

        String path_name = "/user/<Your ID>/lab1/newfile";

        Path path = new Path(path_name);

        // The Output Data Stream to write into
        FSDataOutputStream file = fs.create(path);

        // Write some data
        file.writeChars("The first Hadoop Program!");

        // Close the file and the file system instance
        file.close();
        fs.close();
    }
}
```

The above Java program uses the HDFS Application Programming Interface (API) provided by Hadoop to write into a HDFS file.

Paste the code into your favorite Java editor and change the following:

String path_name = "/user/<Your ID>/lab1/newfile"; to **reflect your actual User ID**.

Note that this is a HDFS path and will not be visible on the local filesystem.

We will transfer this program to the cluster, compile it and then run it to check the output.

You can also compile the program on your local machine if you have the Hadoop libraries downloaded (see the following section on how to compile and run program on hadoop).



Experiment 2 (40 points):

Write a program using the Java HDFS API that reads the contents of the HDFS file `"/cpre419/bigdata"` and computes the 8-bit XOR checksum of all bytes whose offsets range from 1000000000 till 1000000999, both endpoints inclusive. Print out the 8-bit checksum.

Attach the Java code in your submission, as well as the XOR output.

For instance, the XOR checksum of the bitstring `"00000000111111110000000011111111"` is `"00000000"`.

Hint: Use caret '^' to do the XOR.



Transferring Files from/to the Cluster

For Windows:

WinSCP is an open source SFTP and FTP client for Microsoft Windows. It can be used to transfer files securely between a remote and a local machine. You can get the WinSCP client from

<http://winscp.net/eng/index.php>

To connect to the Hadoop Master node:

- Start WinSCP.
- Type the IP address or host name of the Hadoop Master.
- Click **Login**.
- During the first login, you might get a warning message if the server's key is not cached. Click **Yes** to continue.

For UNIX-like systems:

Issue the commands as described below from your local UNIX-like system to transfer files between the Cloud and your local system.

To transfer a file from your local machine to the Hadoop Master Server:

```
$ scp <filename> <user ID>@<server_IP>:/home/<user ID>/<filename>
```

To transfer a file from the server to your local machine:

```
$ scp <user ID>@<server_IP>:/home/<user ID>/<filename> <filename>
```

Using Eclipse IDE with Hadoop

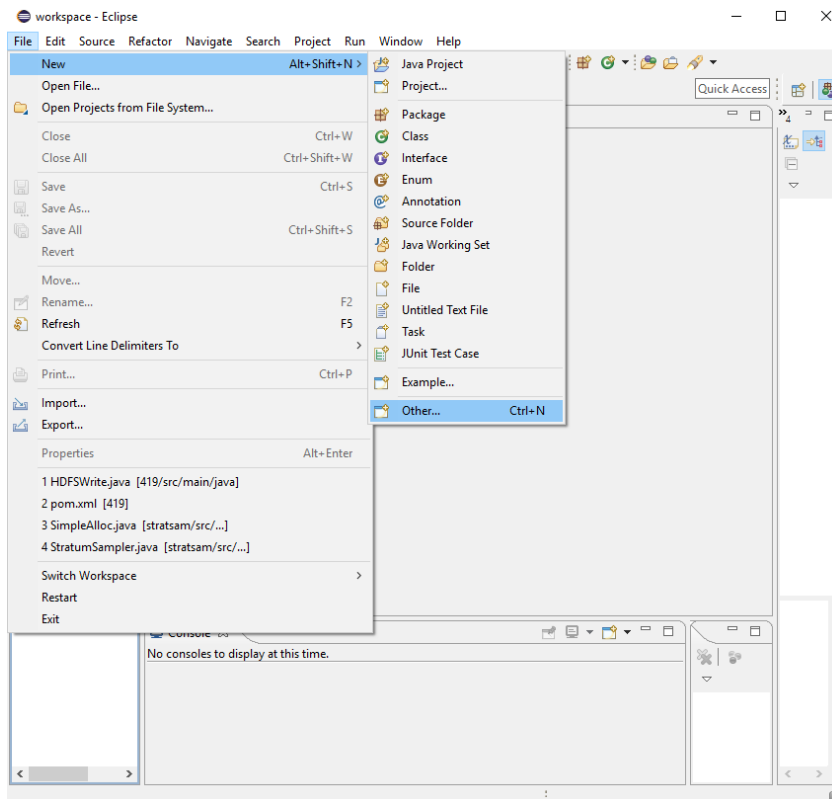
You can compile the Java program manually by the following commands:

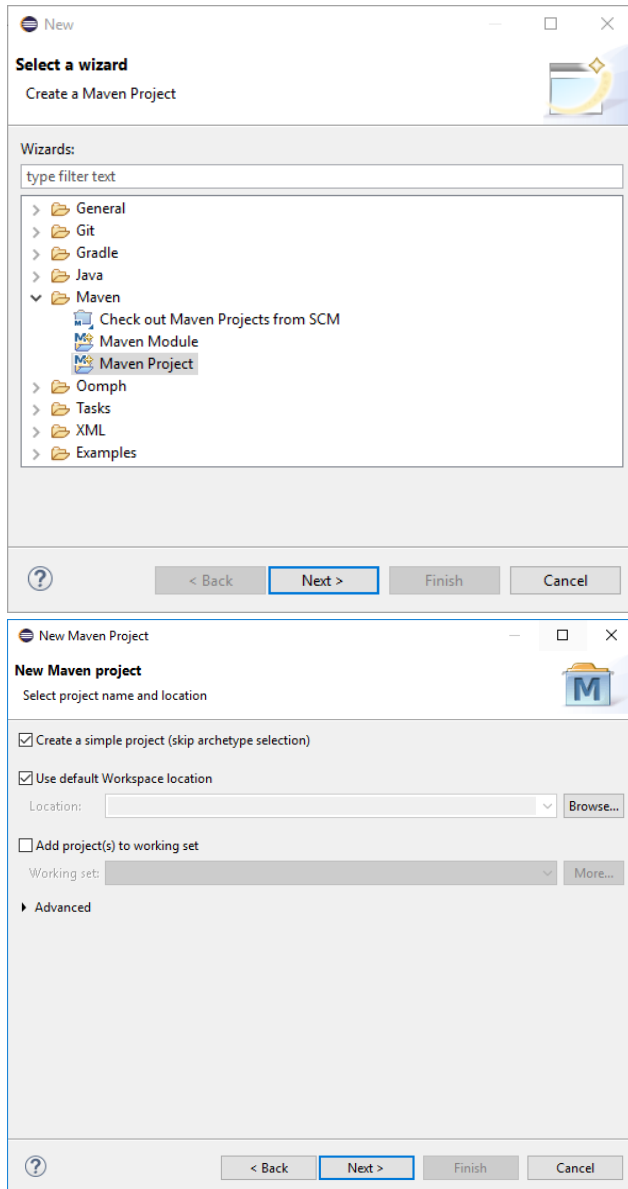
```
$ javac -cp "../libraryfiles/*" -d class/ yourfile.java
$ jar -cvf yourfile.jar -C class/ .
$ rm -rf class
```

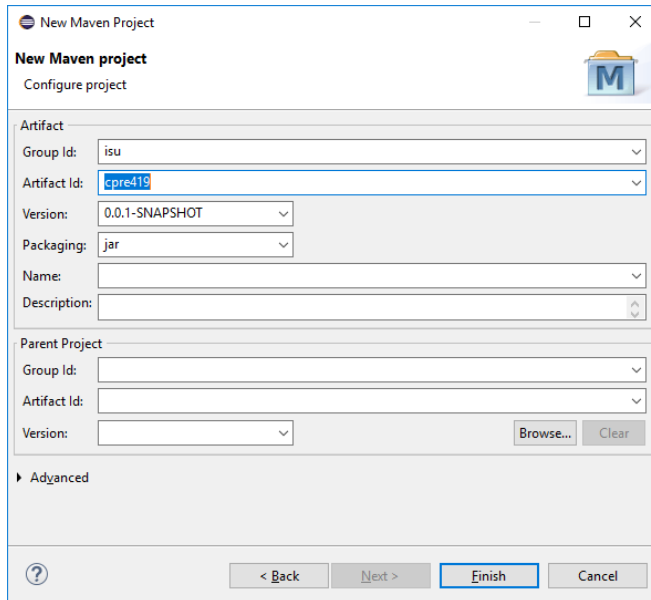
However, using GUI IDE is more efficiency to write your code, compile and debug your program. One of that is Eclipse. Using Eclipse to write your Hadoop program, you need the Hadoop library files. You can manually import those files, which can be found at our course webpage, into your project; Or you can use Maven to automatically import them.

To use Maven:

1. **Open Eclipse IDE**
2. **Create new Maven project**







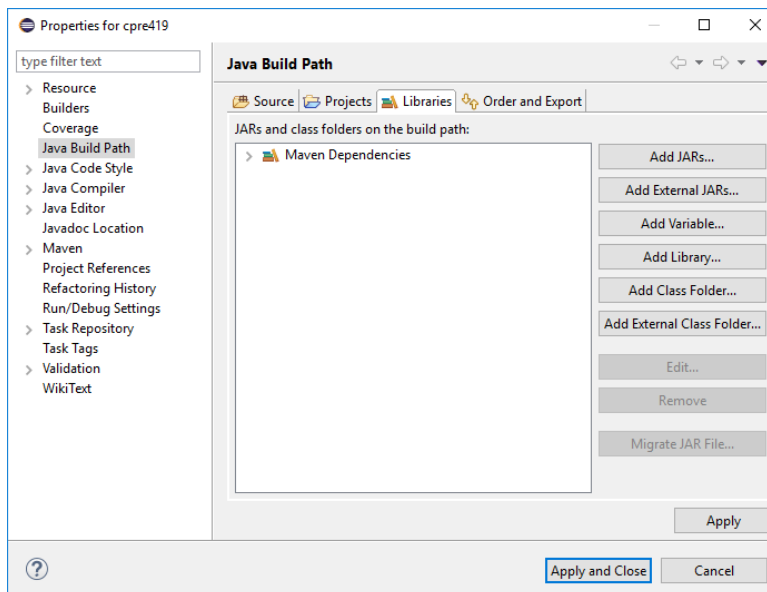
New Maven Project
Configure project

Artifact
Group Id:
Artifact Id:
Version:
Packaging:
Name:
Description:

Parent Project
Group Id:
Artifact Id:
Version:

Advanced

- Right click on the project in the package explorer view. Choose the properties option from the pop-up menu. Once that menu pops up, click on the Java Build Path. Then select libraries tab. You want to have Java 1.8 library, if your current version is something else, remove it. To add the java library, select "Add Library..."



Properties for cpred419

type filter text

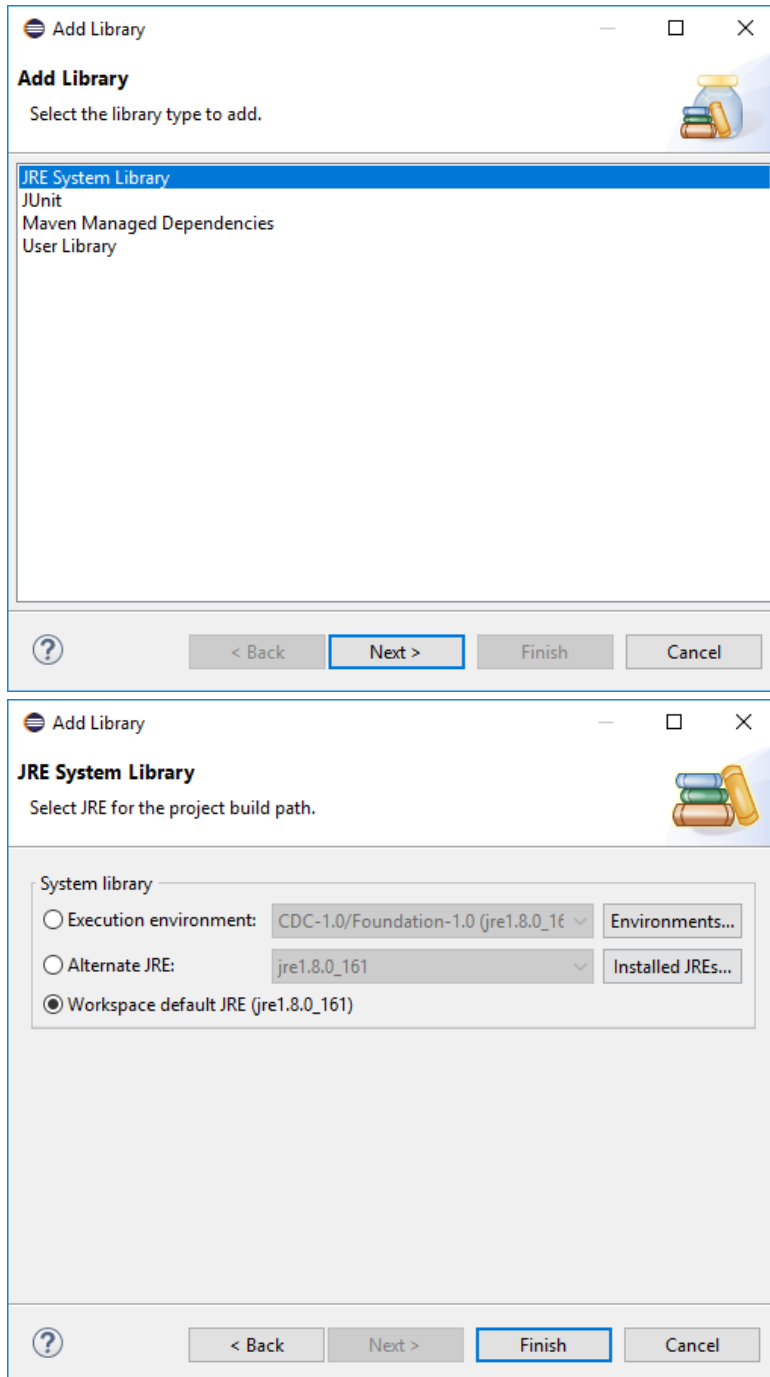
- Resource
- Builders
- Coverage
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- Maven
- Project References
- Refactoring History
- Run/Debug Settings
- Task Repository
- Task Tags
- Validation
- WikiText

Java Build Path

Source Projects Libraries Order and Export

JARs and class folders on the build path:

- Maven Dependencies



4. **Add Hadoop library to maven pom.xml file**
You should have the following in your pom.xml file:



```
...
<dependencies>
  <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common -->
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>2.6.0</version>
    <scope>provided</scope>
  </dependency>

  <dependency>
    <groupId>jdk.tools</groupId>
    <artifactId>jdk.tools</artifactId>
    <version>1.8.0_171</version>
    <scope>system</scope>
    <systemPath>C:/Program Files/Java/jdk1.8.0_171/lib/tools.jar</systemPath>
  </dependency>
</dependencies>
```

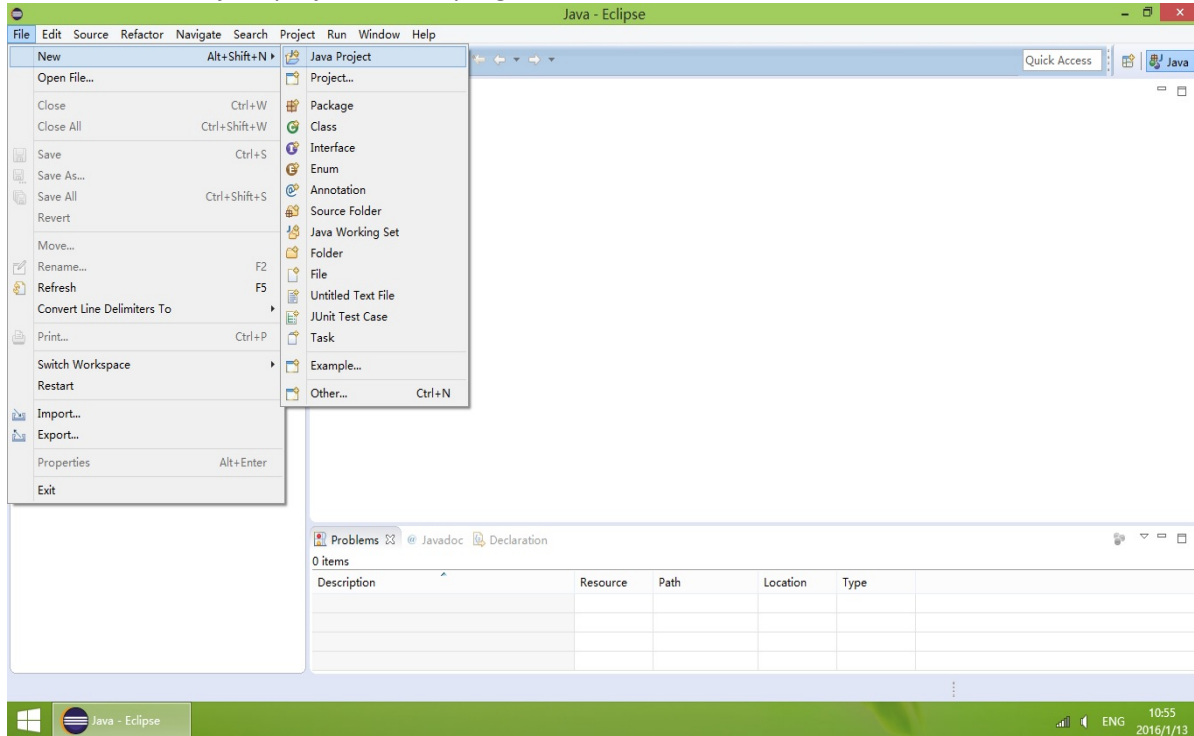
You want to have the following lines in your pom.xml file as well. They make sure that you are using java 1.8

```
<properties>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

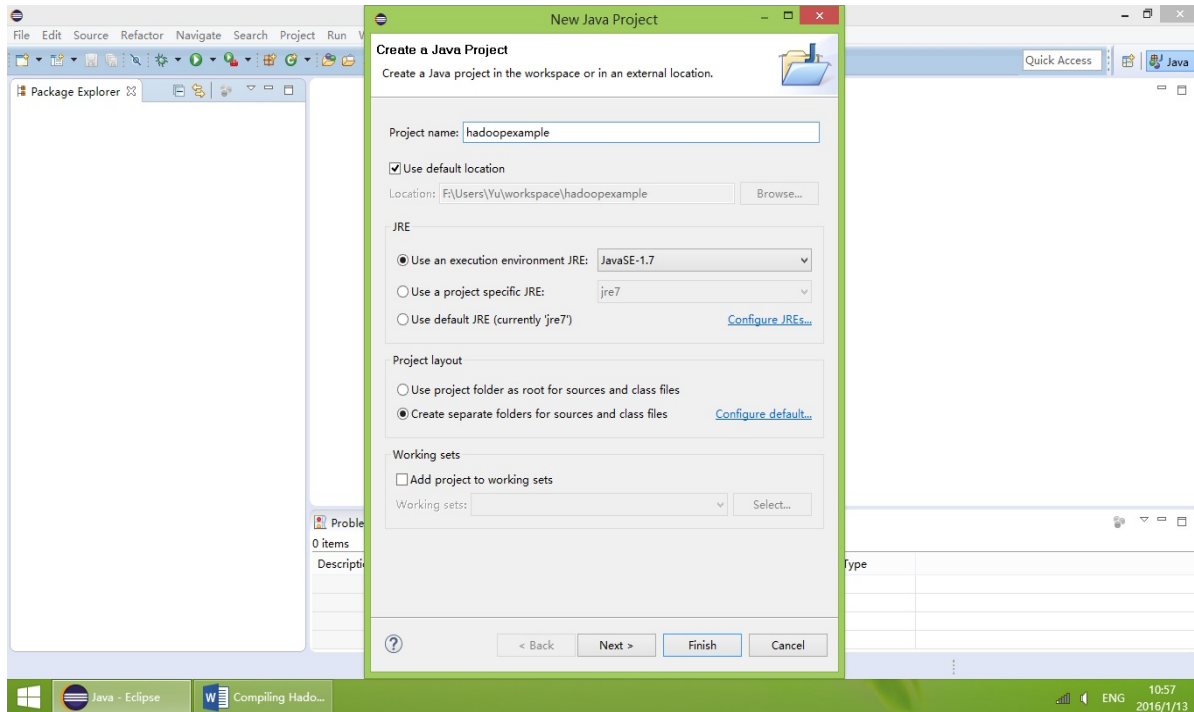


To manually import library files:

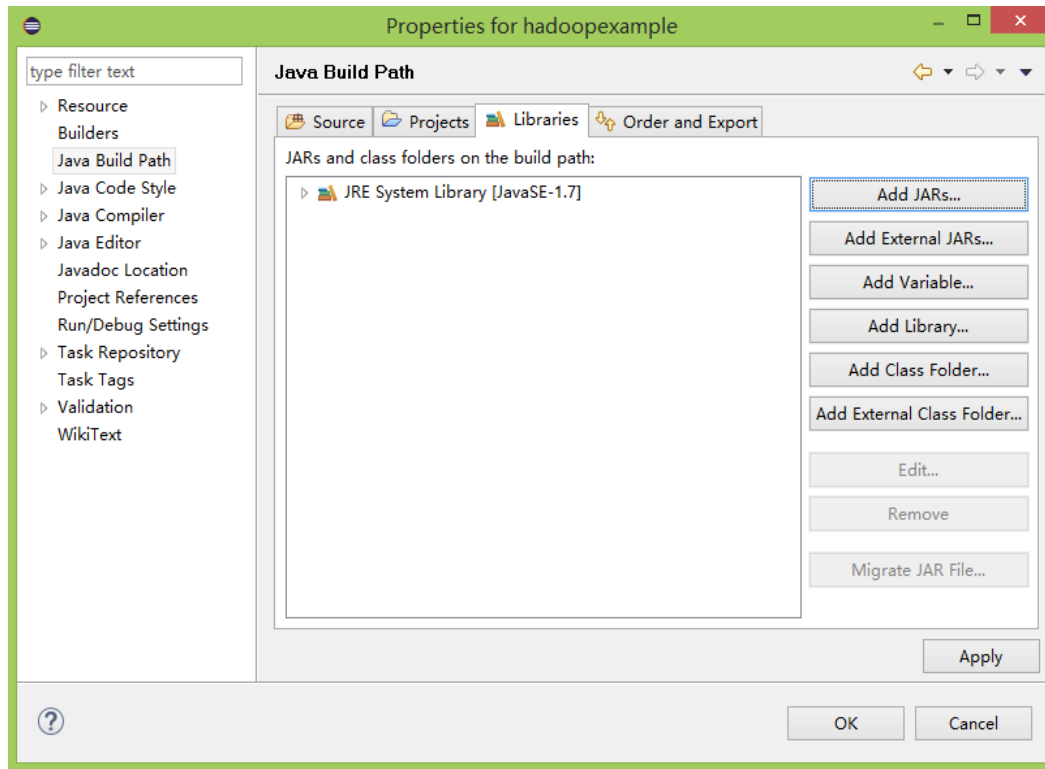
1. Open Eclipse IDE on the lab system.
2. Create a new java project for the program.



3. Enter a name for your new project in the Project Name blank.



4. Right click on the newly created project in the package explorer view. Choose the properties option from the pop-up menu. Once that menu pops up, click on the Java Build Path link. Next, click on the libraries tab.



5. We need to add the hadoop library files into our projects library path. Click on the “Add External Jars” button. The libraries can be found at Canvas Assignment-> Lab1. Download the hadoopjarfiles.zip and unzip it, you will find several jar files in the folder.

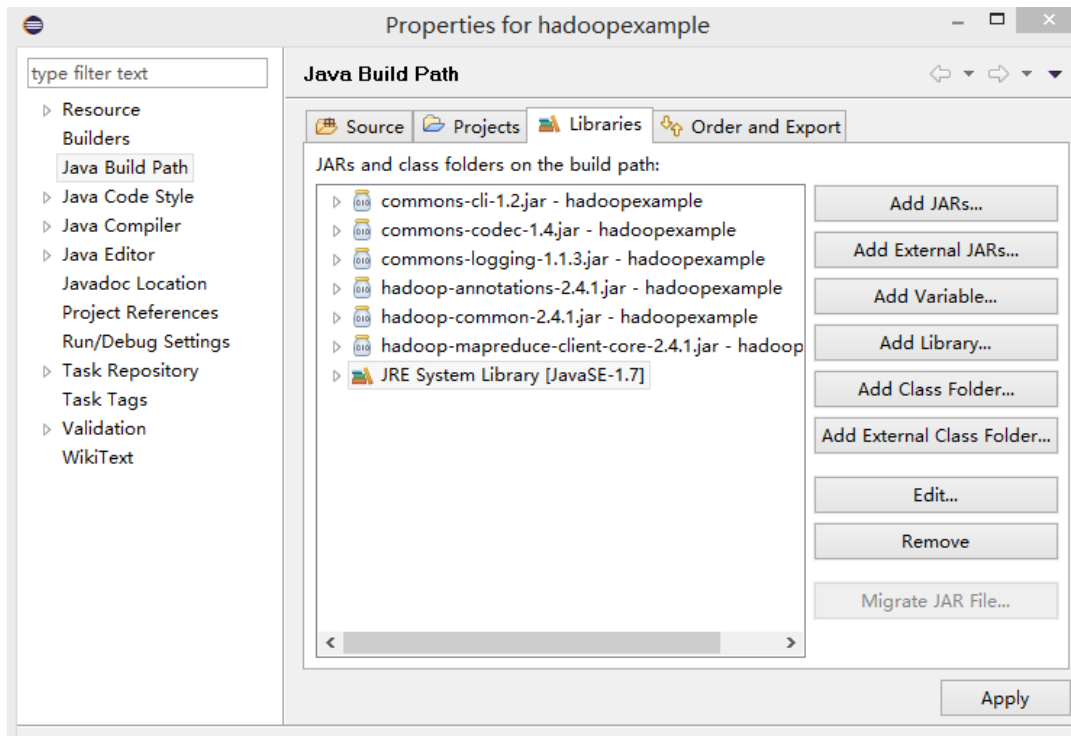
Software to Download

Software to Download	Date
hadoopjarfiles.zip	Jan 15, 2016

commons-cli-1.2.jar	2014/6/21 1:05	Executable Jar File	41 KB
commons-codec-1.4.jar	2014/6/21 1:05	Executable Jar File	57 KB
commons-logging-1.1.3.jar	2014/6/21 1:05	Executable Jar File	61 KB
hadoop-annotations-2.4.1.jar	2014/6/21 1:05	Executable Jar File	17 KB
hadoop-common-2.4.1.jar	2014/6/21 1:05	Executable Jar File	2,784 KB
hadoop-mapreduce-client-core-2.4.1...	2014/6/21 1:05	Executable Jar File	1,459 KB

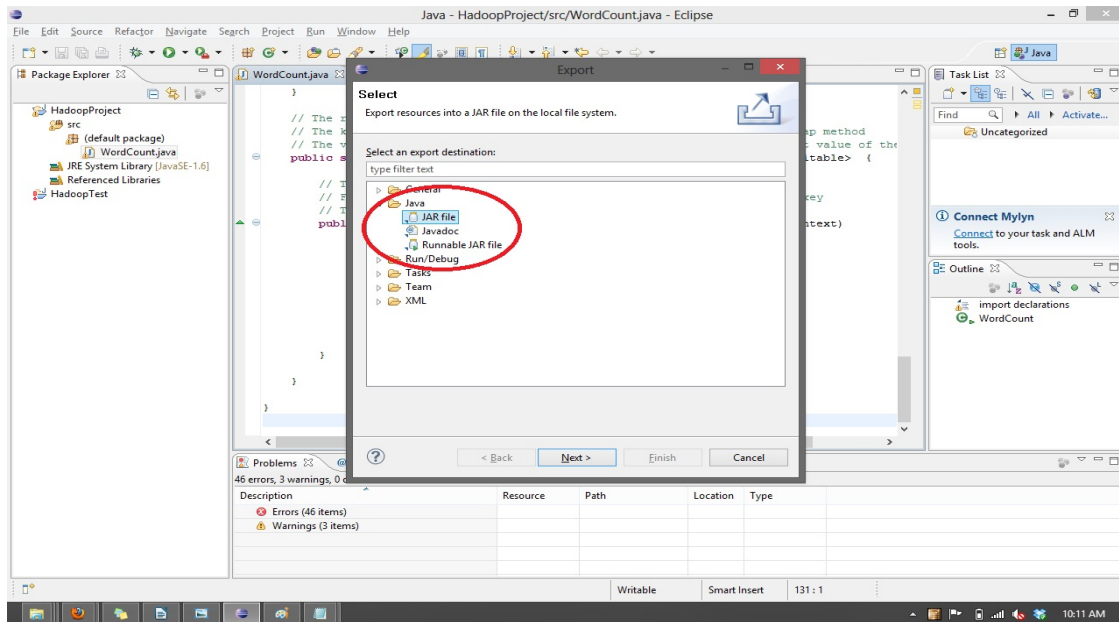


6. Copy all the jar files to your project. Make sure to have all the jar files in the directory added to your project. Once done, press the Ok button and add your source code to the project.

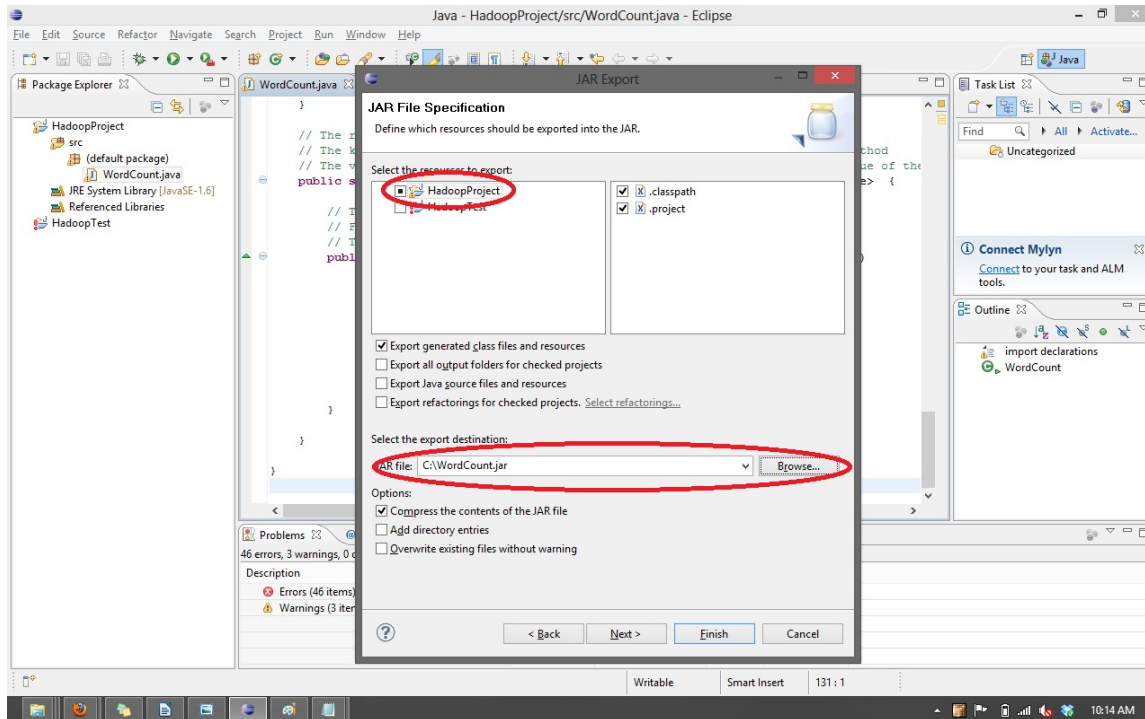


Exporting Jar File

1. Once your code has been added, choose export from the File menu. On the next window, expand the Java section to be able to see the export possibilities for java. Choose the Jar option and press the next button.



2. Check the box by the project that you are wanting to export in the resources window. You will also need to choose a location for the generated jar file to be saved. Then click the finish button to generate the jar file. You may receive a warning about compilation warnings, you may ignore these.



3. You should now have a Jar file at the location you specified in the previous window. This jar file can now be uploaded to the master node of the cluster.

Install Hadoop locally (Optional)

You are encouraged to install Hadoop in your local personal machine. A local version helps you test your solution faster as well as debug it easier.

Hadoop can be download from here: <http://hadoop.apache.org/#Download+Hadoop>

Install on Linux (Ubuntu)

(This guide is tested with Ubuntu 16.04, other version may need minor changes)

Hadoop requires Java be pre-installed. We have Hadoop 2.6.0 and Java 1.8 running in the lab. If you choose different version, just replace the version number in the following commands.

If you don't have java installed, you may use the following command:

```
$ sudo apt-get install default-jdk
```

Checking version of java:



```
$ java -version
```

You can download Hadoop from Apache Hadoop Releases page, look like so:

Apache > Hadoop >

APACHE hadoop

Search with Apache Solr Search

Last Published: 12/18/2017 07:18:32

Apache Hadoop Releases

Download

Hadoop is released as source code tarballs with corresponding binary tarballs for convenience. The downloads are distributed via mirror sites and should be checked for tampering using GPG or SHA-256.

Version	Release Date	Tarball	GPG	SHA-256
3.0.0	13 December, 2017	source	signature	checksum file
		binary	signature	checksum file
2.9.0	17 November, 2017	source	signature	checksum file
		binary	signature	checksum file
2.8.3	12 December, 2017	source	signature	checksum file
		binary	signature	checksum file
2.7.5	14 December, 2017	source	signature	D7523A9E D8FD404C...
		binary	signature	0BFC4D9B 048E919B...
2.6.5	08 October, 2016	source	signature	3A843F18 73D9951A...
		binary	signature	001AD18D 4B6D0FE5...

To verify Hadoop releases using GPG:

1. Download the release `hadoop-X.Y.Z-src.tar.gz` from a [mirror site](#).
2. Download the signature file `hadoop-X.Y.Z-src.tar.gz.asc` from [Apache](#).
3. Download the [Hadoop KEYS](#) file.
4. `gpg --import KEYS`
5. `gpg --verify hadoop-X.Y.Z-src.tar.gz.asc`

To perform a quick check using SHA-256:

1. Download the release `hadoop-X.Y.Z-src.tar.gz` from a [mirror site](#).
2. Download the checksum `hadoop-X.Y.Z-src.tar.gz.md5` from [Apache](#).
3. `shasum -a 256 hadoop-X.Y.Z-src.tar.gz`

All previous releases of Hadoop are available from the [Apache release archive](#) site.

Many third parties distribute products that include Apache Hadoop and related tools. Some of these are listed on the [Distributions wiki page](#).

You also have option to use `wget` to fetch the file:

```
$ wget http://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-2.6.0/hadoop-2.6.0.tar.gz
```

You can run the verification with checksum, then extract the compressed file:

```
$ tar -xzf hadoop-2.6.0.tar.gz
```

Finally, we'll move the extracted files into `/usr/local`, the appropriate place for locally installed software. Change the version number, if needed, to match the version you downloaded.

```
$ sudo mv hadoop-2.6.0 /usr/local/Hadoop
```

Hadoop requires that you set the path to Java, either as an environment variable or in the Hadoop configuration file.

The path to Java, `/usr/bin/java` is a symlink to `/etc/alternatives/java`, which is in turn a symlink to default Java binary. We will use `readlink` with the `-f` flag to follow every symlink in every part of the path, recursively. Then, we'll use `sed` to trim `bin/java` from the output to give us the correct value for `JAVA_HOME`.



To find the default Java path:

```
$ readlink -f /usr/bin/java | sed "s:bin/java::"
```

You can copy the output to set Hadoop's Java home to this specific version. Alternatively, you can use the readlink command dynamically in the file so that Hadoop will automatically use whatever Java version is set as the system default.

The configure of Hadoop is in the file: `/usr/local/hadoop/etc/hadoop/hadoop-env.sh`

You should have something like so:

```
. . .
#export JAVA_HOME=${JAVA_HOME}
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre/
. . .
```

Done!

You can give a try now

```
$ /usr/local/hadoop/bin/hadoop
```

You should see something like so:

```
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
  credential               interact with credential providers
                           Hadoop jar and the required libraries
  daemonlog               get/set the log level for each daemon
```