

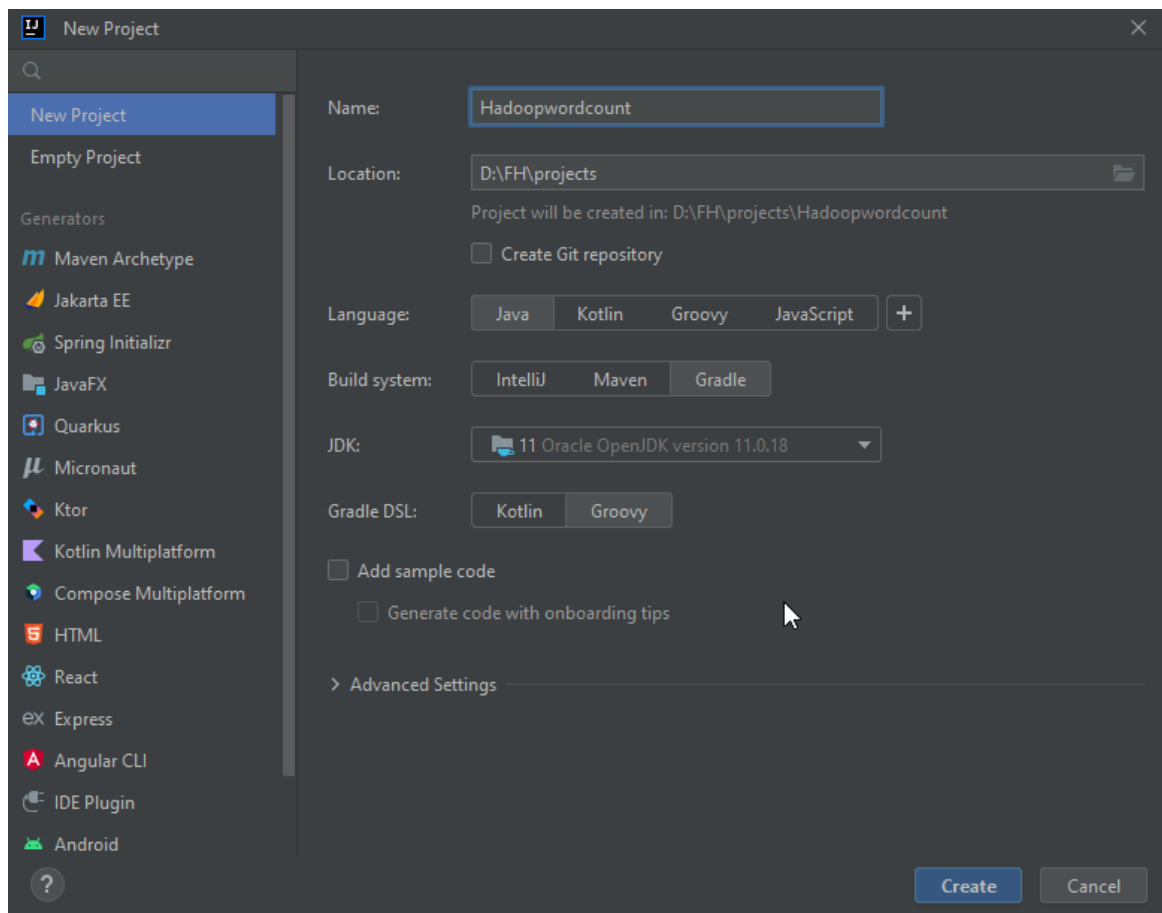
MapReduce job mit Java in IntelliJ IDE

IntelliJ community Edition (lizenzfrei) von

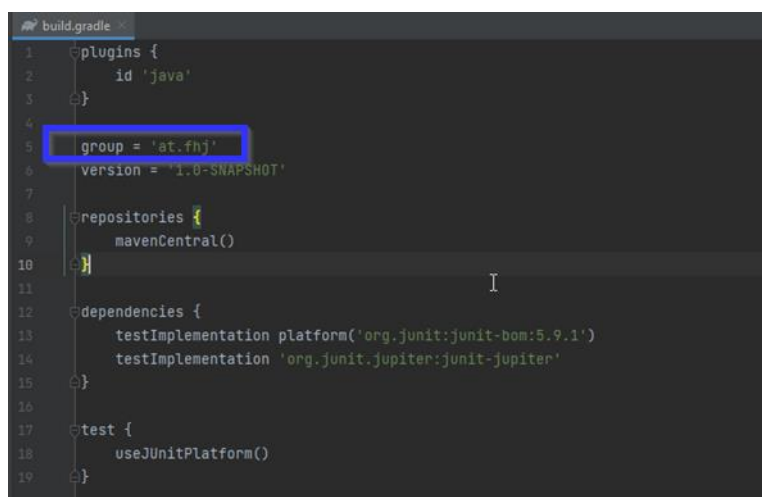
<https://www.jetbrains.com/idea/download/?section=windows> herunterladen

Nach Installation ein neues Java-Projekt anlegen, passende JDK-Version auswählen.

Am besten gradle oder maven als Buildsystem auswählen, damit dies leichter wiederverwendbar ist.

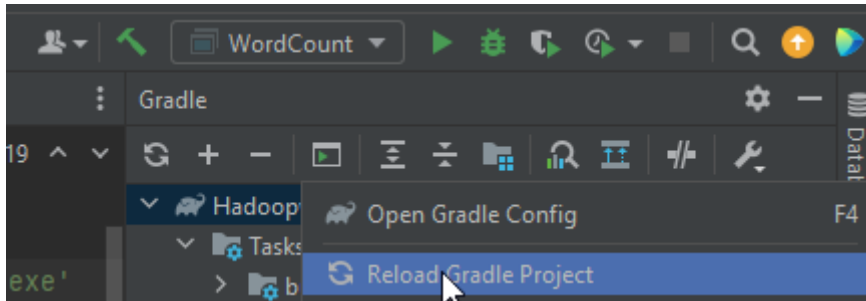


das automatisch generierte build.gradle sollte wie folgt aussehen, nur die "group" ist zu ändern.

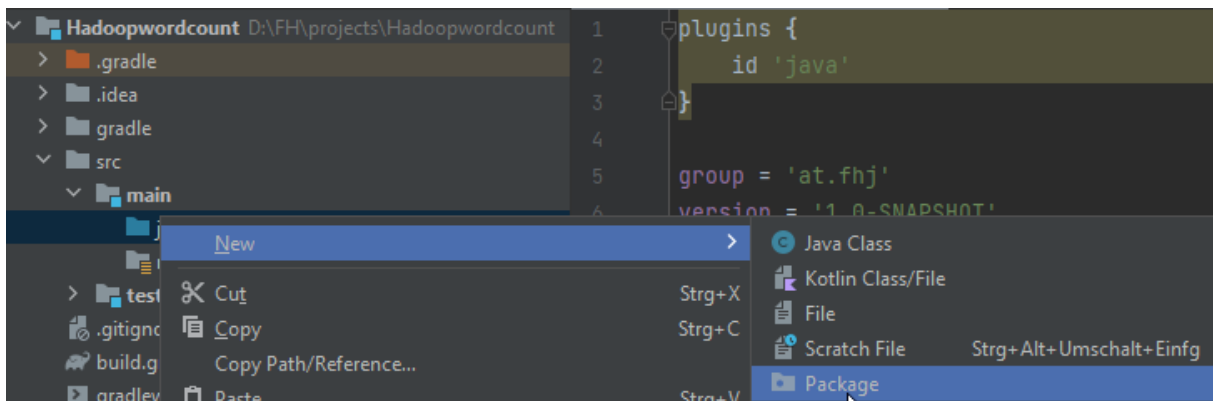


Besser jedoch das build.gradle aus dem github-Repo

(<https://github.com/Woberegger/BigData/tree/main/src/wordcount>) abholen, dort sind auch die dependencies schon eingetragen – danach folgendes ausführen, damit das build-File neu geladen wird – es sollten darauf hin etliche Jar-Files aus dem zentralen Repo im Internet runtergeladen werden:

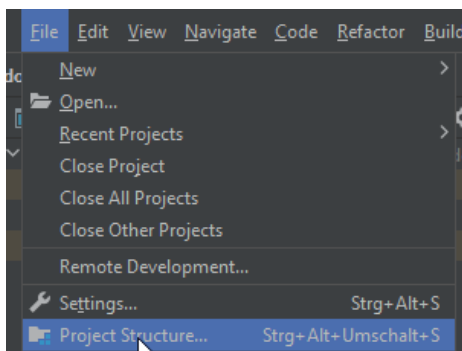


ein neues Package at.fhj anlegen, damit die Struktur der Defaultstruktur entspricht.



Dort die 3 Sourcefiles ablegen (am einfachsten durch Kopieren von <https://github.com/Woberegger/BigData/tree/main/src/wordcount> in das zuvor angelegte Package-Directory)

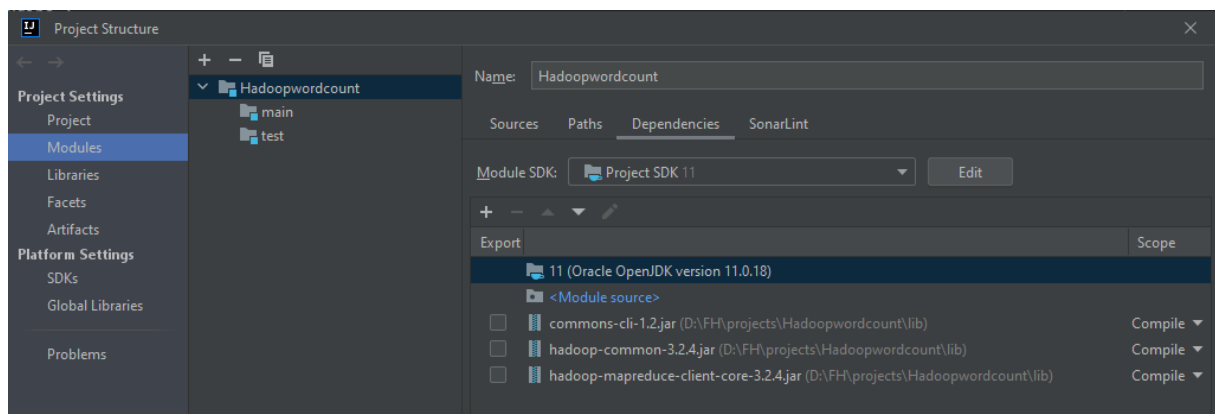
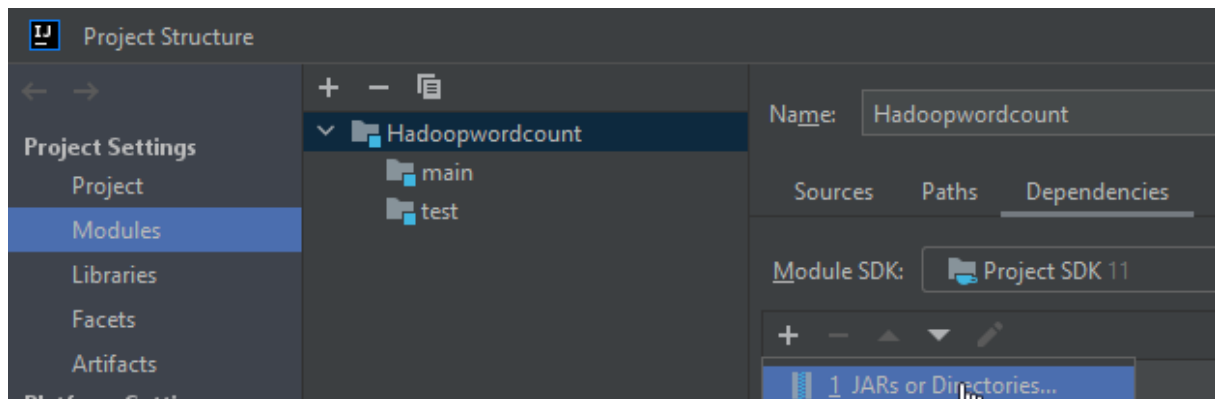
Projektstruktur anpassen:



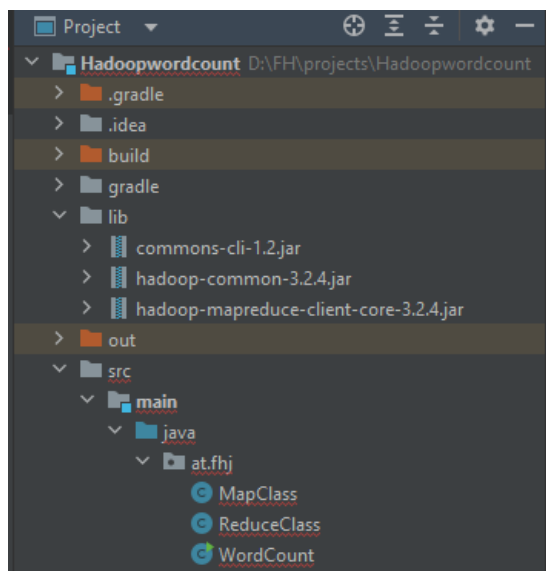
Damit die Dependencies aufgelöst werden, gibt es mehrere Möglichkeiten:

- a) MavenCentral repo verwenden (siehe build.gradle)

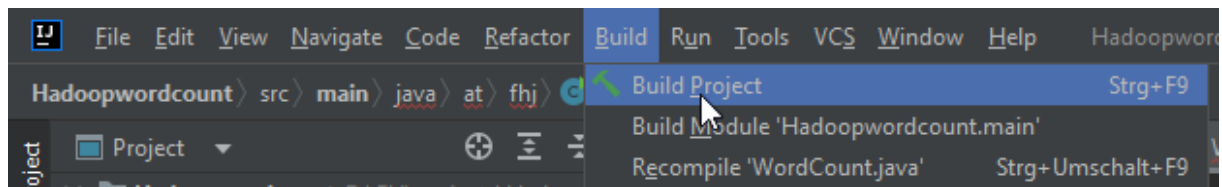
- b) Jar-Dateien zum Projekt hinzufügen (die Dateien entweder aus der E-Learning-Plattform oder noch besser vom installierten hadoop in der virtuellen Maschine – ACHTUNG: Versionen müssen zusammenpassen, es ist aber möglich, mit niedrigerer Version zu bauen, als die Version in der virtuellen Maschine):



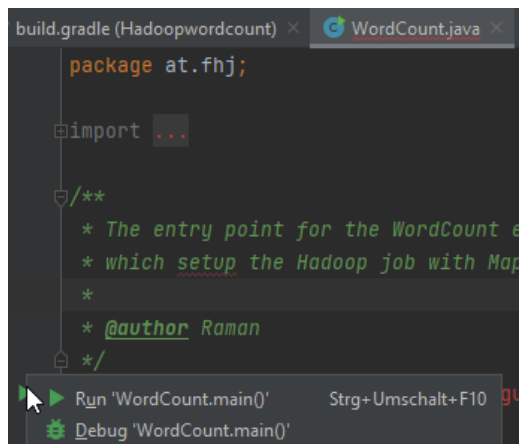
Projektstruktur, nachdem die Sourcedateien dort abgelegt wurden



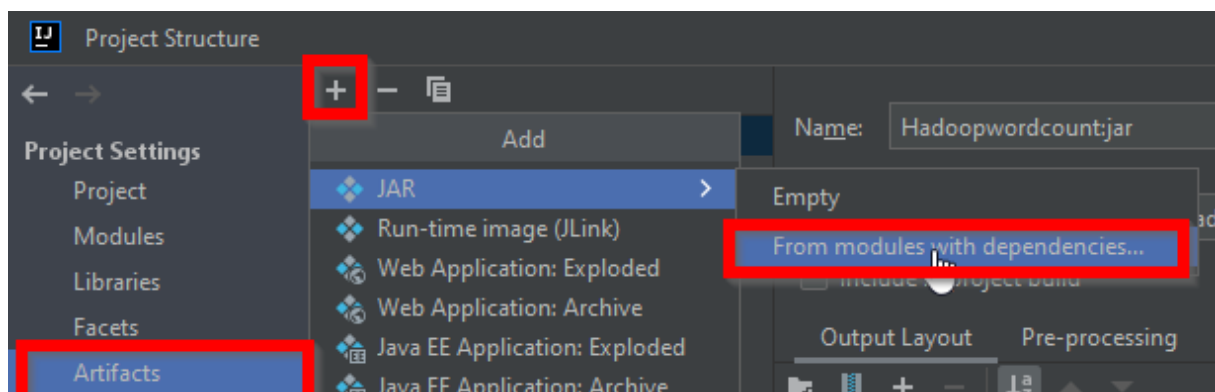
Projekt bauen, d.h. kompilieren

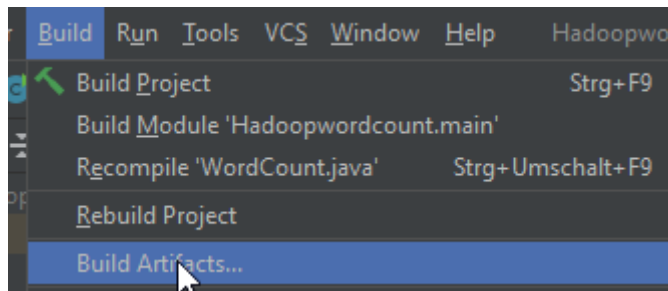
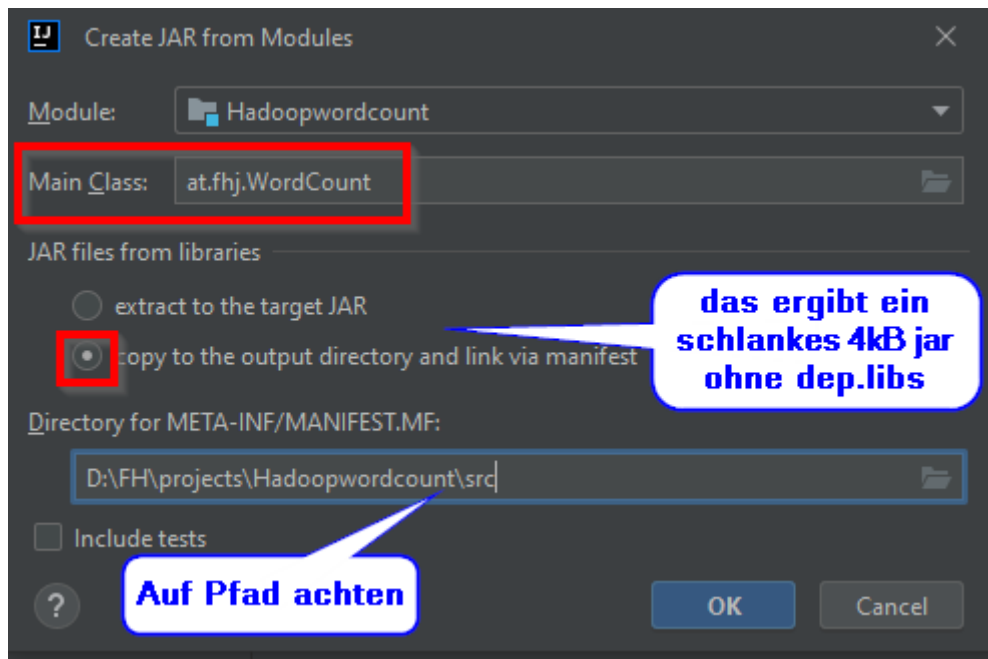


Ausführen oder Debuggen direkt bei den jeweiligen Methoden (müsste man remote debuggen auf der Zielplattform verwenden, wo DFS installiert ist), d.h. dieser Schritt ist im Normalfall zu überspringen.



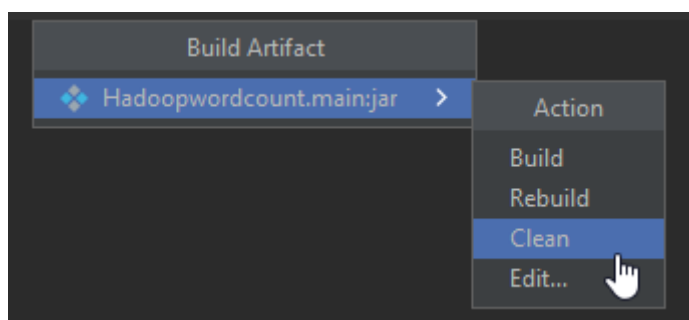
Artifakt anlegen, d.h. über "Project structure" das Erstellen des jar-File konfigurieren und dem Jar-File mitteilen, wo der Einsprungspunkt, d.h. die Main-Klasse, zu finden ist.





Je nach verwendetem Buildsystem und deren Settings findet man das generierte Jar-File unter z.B. out/artifacts oder build/libs und beinhaltet der Dateiname eine Versionsnr. oder nicht.

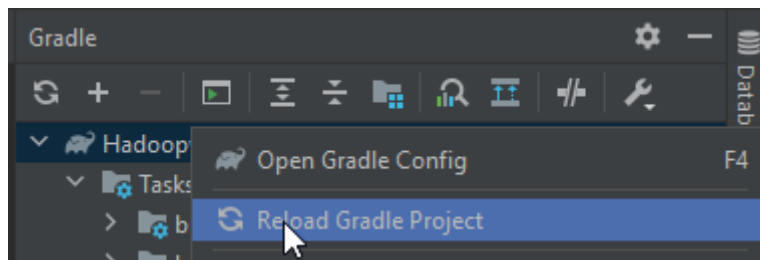
Falls zuvor schon eine Version dort gebildet wurde, die nicht funktioniert (weil z.B. fehlende Dependencies), diese besser über "Clean" zuerst löschen.



Falls auf dem lokalen System eine höhere Java-Version ("java -version") installiert ist als in der virtuellen Maschine, dann sollte folgende Einstellung ins build.gradle übernommen werden (optional Version VERSION_1_8 verwenden, falls dies verlangt wird – kann gerade bei MacOS nötig sein).

```
java {
    sourceCompatibility = JavaVersion.VERSION_11
    targetCompatibility = JavaVersion.VERSION_11
}
```

Wenn man gradle verwendet mit einem reinkopierten build.gradle, sollte man Folgendes ausführen:



Optional auf Kommandozeile:

```
gradle task Jar
```

Hadoop starten und Test aufrufen mit dem zuvor generierten und auf die Zielplattform kopierten Jar-File.

Job ausführen

- 1.) Input Ordner und Beispiel Textdatei im hdfs anlegen
- 2.) Job ausführen (Vorsicht Pfade sind ggf. anzupassen!)

```
su - hduser
start-dfs.sh
start-yarn.sh
hdfs dfs -mkdir /input
hdfs dfs -mkdir /output
hdfs dfs -put ~/BigData/data/Bibel.txt /input/
OutputDir=/output/Bibel
# folgendes für jeden Versuch neu ausführen
hdfs dfs -rm -R $OutputDir
hadoop jar HadoopWordCount.jar /input/Bibel.txt $OutputDir
```

Bei Problemen die Jar-Datei überprüfen ("jar -tvf" funktioniert meines Wissens nur unter Linux), sie muss zumindest folgende Dateien beinhalten:

```
jar -tvf HadoopWordCount.jar | awk '{ print $8 }'
META-INF/MANIFEST.MF
at/fhj/MapClass.class
at/fhj/ReduceClass.class
at/fhj/WordCount.class
```

Der Inhalt der Datei MANIFEST.MF sollte wie folgt sein:

```
jar -xf minimal.jar META-INF/MANIFEST.MF
cat META-INF/MANIFEST.MF
Manifest-Version: 1.0
Main-Class: at.fhj.WordCount
```

Erwarteter Output nach erfolgreichem MapReduce Job wie folgt:

```
hduser@osboxes: ~/Desktop

Reduce input groups=56030
Reduce shuffle bytes=9199249
Reduce input records=781853
Reduce output records=56030
Spilled Records=1563706
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=55
Total committed heap usage (bytes)=706740224

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=4579278
File Output Format Counters
  Bytes Written=614705

Job was successful
hduser@osboxes:~/Desktop$
```

Im angegebenen Output-Verzeichnis befinden sich dann 2 Dateien, eine leere Datei "_SUCCESS" und eine Datei mit dem Ergebnis des Jobs.

localhost:9870/explorer.html#/output/bibel 110%

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory

/output/bibel Go! [Icons]

Show 25 entries Search:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	hduser	supergroup	0 B	Nov 04 15:18	2	16 MB	_SUCCESS	
<input type="checkbox"/>	-rw-r--r--	hduser	supergroup	363.59 KB	Nov 04 15:18	2	16 MB	part-r-00000	

File information - part-r-00000

[Download](#)[Head the file \(first 32K\)](#)[Tail the file \(last 32K\)](#)

Block information -- Block 0

Block ID: 1073741956

Block Pool ID: BP-109114966-127.0.1.1-1698821495461

Generation Stamp: 1138

Size: 372320

Availability:

- UbuntuBigData

File contents

```
mme      1
verstummen  1
verstummt  2
verstumnte  1
```

Wichtig: wenn der Job erneut gestartet werden soll, muss zuvor das Output-Verzeichnis gelöscht werden (sowohl wenn man Output ins lokale Dateisystem als auch wenn man dies in hdfs schreibt)!