

# Instalacion, Configuracion y Documentacion de Spark

Univ. Sergio Alejandro Paucara Saca

12 de diciembre de 2022

## Índice

<b>1. Instalacion Spark</b>	<b>2</b>
1.1. Actualizacion paquetes Linux - Ubuntu . . . . .	2
1.2. Instalación Java Runtime . . . . .	2
1.3. Descargar e Instalacion de Apache Spark . . . . .	3
1.3.1. Configuracion Variables de Entorno - Spark . . . . .	4
1.4. Iniciar a standalone master server . . . . .	4
1.5. Iniciar Spark Worker Process . . . . .	5
1.6. Usar Spark desde la terminal . . . . .	6
1.6.1. Scala para Spark . . . . .	6
1.6.2. Python para Spark . . . . .	6
<b>2. Apache Spark - DocumentacionCodigo</b>	<b>7</b>
2.1. Sesiones . . . . .	7
2.1.1. Codigo Refactorizado . . . . .	7
2.2. Streaming . . . . .	8
2.2.1. Codigo Refactorizado . . . . .	8
2.3. RDD . . . . .	8
2.3.1. Codigo Refactorizado . . . . .	8

# Índice de cuadros

## 1. Instalacion Spark

En una máquina virtual realice la configuración de apache spark, puede guiarse en cualquier tutorial o el proporcionado por el docente. url: <https://computingforgeeks.com/how-to-install-apache-spark-on-ubuntu-debian/>  
Con el shell podra ejecutar scala por defecto  
Instale Python para spark

### 1.1. Actualizacion paquetes Linux - Ubuntu

Previo a la instalacion de Spark, se tiene que tener instalado el sistema operativo Ubuntu con preferencia en la version 22.04.1 de 64 bits. La instalacion puede ser en una maquina virtual.

Ingresar a la terminal del sistema operativo Ubuntu 22.04.1, logearse como usuario root con la siguiente linea de comando:

```
$ sudo su
```

Nos pedira la contraseña con la cual configuramos el sistema operativo. Luego debemos actualizar todo los paquetes del sistema con la siguiente linea de comando:

```
$ sudo apt update && sudo apt -y full-upgrade
```

La actualización demorara unos minutos.

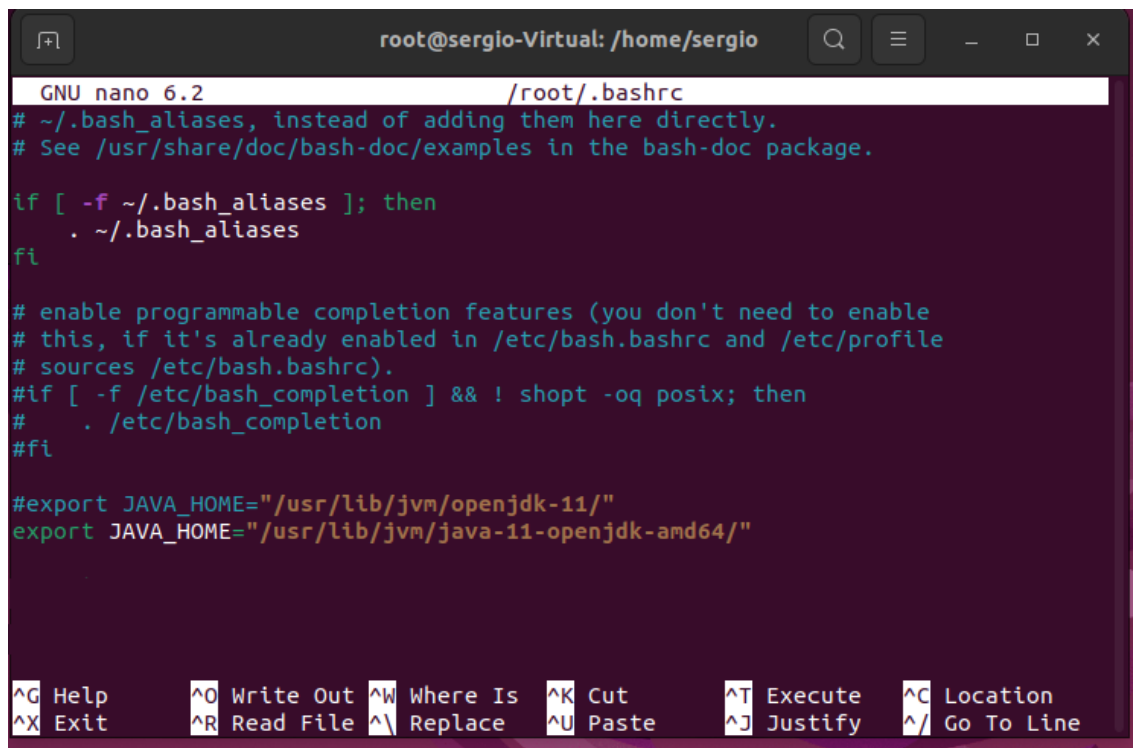
### 1.2. Instalación Java Runtime

Como Apache Spark requiere Java, es necesario realizar la instalacion del programa., para ello se tiene que seguir las siguientes lineas de comando en la terminal de Ubuntu:

```
$ sudo apt update  
$ java -version  
$ sudo apt install default-jre  
$ java -version
```

Abrimos el archivo **bashrc** agreamos la siguiente linea al final del archivo y guardamos.

```
$ nano ~/.bashrc  
export JAVA_HOME="/usr/lib/jvm/java-11openjdk-amd64/"
```



```
GNU nano 6.2 /root/.bashrc
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

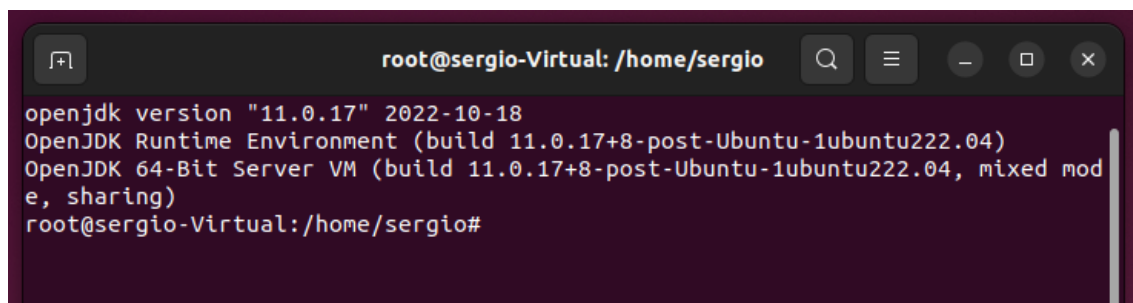
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
#if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
#    . /etc/bash_completion
#fi

#export JAVA_HOME="/usr/lib/jvm/openjdk-11/"
export JAVA_HOME="/usr/lib/jvm/java-11-openjdk-amd64/"

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute  ^C Location
^X Exit      ^R Read File ^_ Replace   ^U Paste     ^J Justify  ^_ Go To Line
```

Finalmente para activar los cambios, ejecutamos la siguiente linea:

```
$ source ~/.bashrc
```



```
openjdk version "11.0.17" 2022-10-18
OpenJDK Runtime Environment (build 11.0.17+8-post-Ubuntu-1ubuntu222.04)
OpenJDK 64-Bit Server VM (build 11.0.17+8-post-Ubuntu-1ubuntu222.04, mixed mode, sharing)
root@sergio-Virtual:/home/sergio#
```

### 1.3. Descargar e Instalacion de Apache Spark

Descargamos la ultima version de Apache Spark - 3.3.1, con el comando **wget**:

```
$ wget https://dlcdn.apache.org/spark/spark-3.3.1/spark-3.2.1-bin-hadoop3.tgz
```

Descomprimos el archivo **tar** que descargamos:

```
$ tar xvf spark-3.3.1-bin-hadoop3.tgz
```

Después de descomprimir el archivo se nos creara una carpeta la cual la tenemos que mover al directorio **/opt/**

```
$ sudo mv spark-3.3.1-bin-hadoop3/ /opt/spark
```

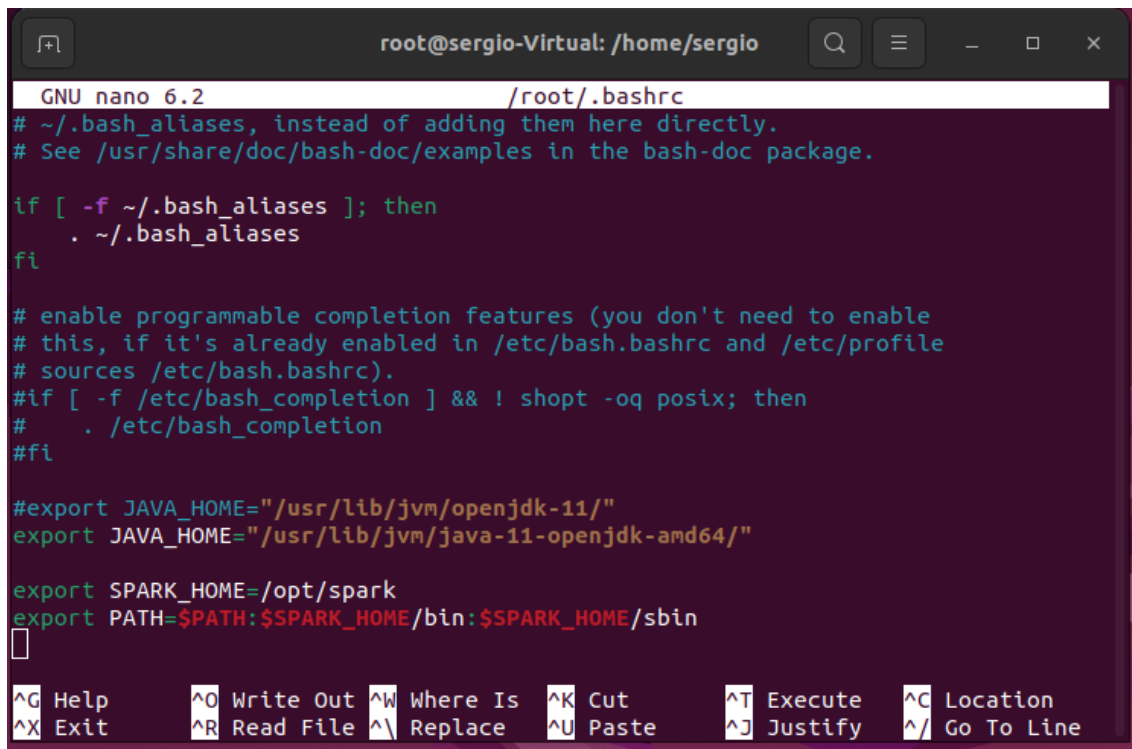
### 1.3.1. Configuración Variables de Entorno - Spark

Listo lo tenemos ya instalado, ahora se tiene que configurar el entorno de las variables de Spark, abrimos **bashrc**

```
$ nano ~/.bashrc
```

Y al final del archivo agregamos las siguientes líneas:

```
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
```



```
GNU nano 6.2 /root/.bashrc
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
#if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
#    . /etc/bash_completion
#fi

#export JAVA_HOME="/usr/lib/jvm/openjdk-11/"
export JAVA_HOME="/usr/lib/jvm/java-11-openjdk-amd64/"

export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin

```

Finalmente para activar los cambios, ejecutamos la siguiente línea:

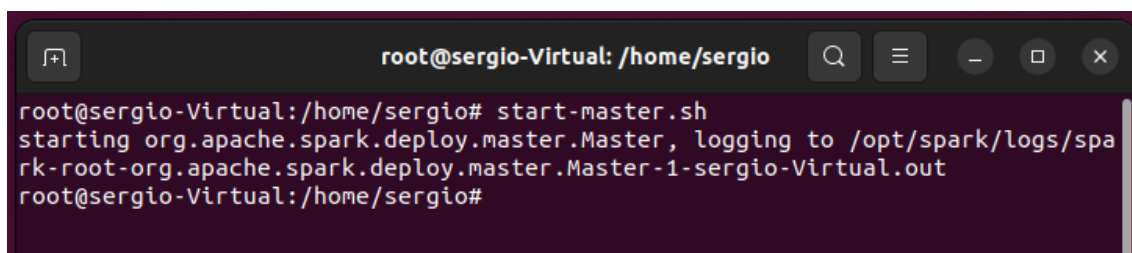
```
$ source ~/.bashrc
```

Y ya tenemos instalado Apache Spark para Scala y Python

### 1.4. Iniciar a standalone master server

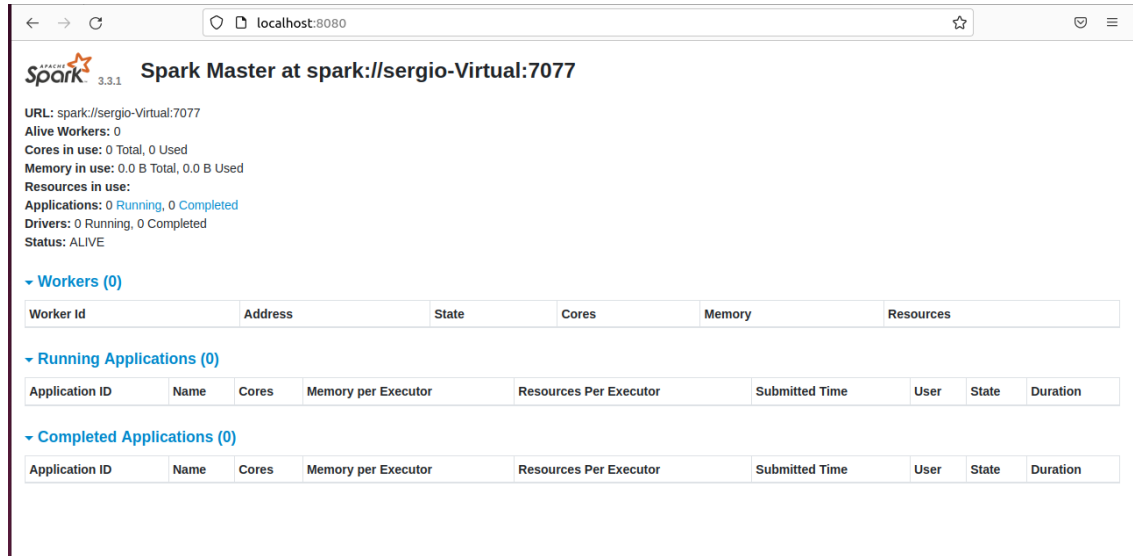
Para iniciar es necesario poner la siguiente línea de comando:

```
$ start-master.sh
```



```
root@sergio-Virtual: /home/sergio
root@sergio-Virtual:/home/sergio# start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark-root-org.apache.spark.deploy.master.Master-1-sergio-Virtual.out
root@sergio-Virtual:/home/sergio#
```

Y si entramos desde un navegador para ver su UI podemos ver:



Spark Master at spark://sergio-Virtual:7077

URL: spark://sergio-Virtual:7077  
Alive Workers: 0  
Cores in use: 0 Total, 0 Used  
Memory in use: 0.0 B Total, 0.0 B Used  
Resources in use:  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

Workers (0)

Worker Id	Address	State	Cores	Memory	Resources
-----------	---------	-------	-------	--------	-----------

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

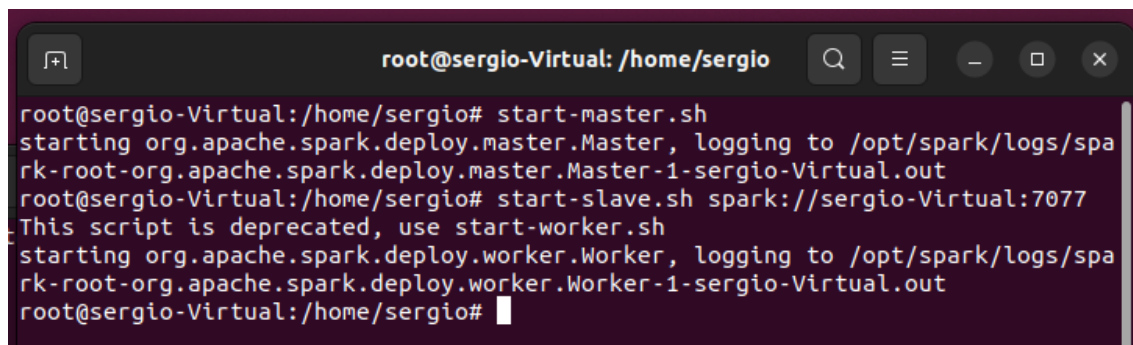
Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

## 1.5. Iniciar Spark Worker Process

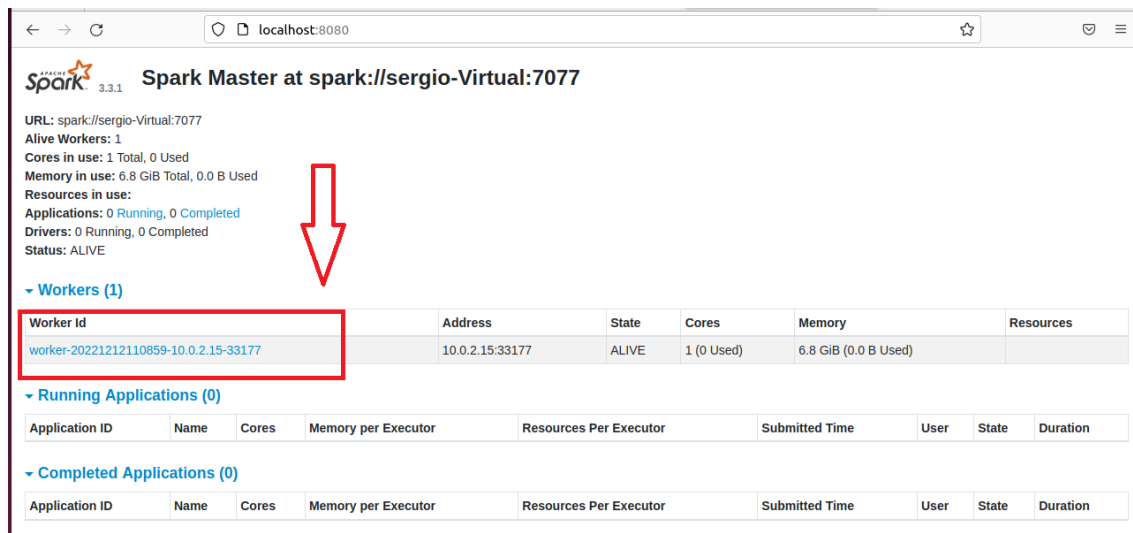
El comando para arracar los esclavos nos ayuda a iniciar Spark Worker Process, en este caso my URL de Spark es `spark://sergio-Virtual:7077`

```
$ start-slave.sh spark://sergio-Virtual:7077
```



```
root@sergio-Virtual: /home/sergio
root@sergio-Virtual:/home/sergio# start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark-root-org.apache.spark.deploy.master.Master-1-sergio-Virtual.out
root@sergio-Virtual:/home/sergio# start-slave.sh spark://sergio-Virtual:7077
This script is deprecated, use start-worker.sh
starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-sergio-Virtual.out
root@sergio-Virtual:/home/sergio#
```

Podemos observar en la UI de Spark, que se inicio un Spark Worker Process



Spark Master at spark://sergio-Virtual:7077

URL: spark://sergio-Virtual:7077  
Alive Workers: 1  
Cores in use: 1 Total, 0 Used  
Memory in use: 6.8 GiB Total, 0.0 B Used  
Resources in use:  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory	Resources
worker-20221212110859-10.0.2.15-33177	10.0.2.15:33177	ALIVE	1 (0 Used)	6.8 GiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------



## 2. Apache Spark - DocumentacionCodigo

Realice el siguiente código, documente su funcionamiento en apache spark

### 2.1. Sesiones

```
val spark: SparkSession = SparkSession.builder()  
  .master("local[*]")  
  .appName("simple-app")  
  .getOrCreate()  
  
val dataSet: Dataset[String] = spark.read.textFile("textfile.csv")  
val df: DataFrame = dataSet.toDF()
```

Previo a ejecutar el código se tiene que hacer una refactorización ya que falta importar algunas librerías y también es necesario crear un archivo csv.

#### 2.1.1. Código Refactorizado

Se debe importar las siguientes librerías.

```
import org.apache.spark.sql.SparkSession  
import org.apache.spark.sql.Dataset  
import org.apache.spark.sql.DataFrame
```

La siguiente línea nos ayuda a crear una sesión en Scala con Spark

```
val spark: SparkSession = SparkSession.builder()  
  .master("local[*]")  
  .appName("simple-app")  
  .getOrCreate()
```

Para la lectura de un DataSet en formato de CSV.

```
val dataSet: Dataset[String] = spark.read.textFile("textfile.csv")
```

Para convertir el DataSet que leímos en formato CSV a un DataFrame.

```
val df: DataFrame = dataSet.toDF()  
df.show()
```

## 2.2. Streaming

```
val streamingContext: StreamingContext = new StreamingContext(sparkContext, Seconds(20))
val lines: ReceiverInputDStream[String] = streamingContext.socketTextStream("localhost",
  ↳ 9999)
```

### 2.2.1. Código Refactorizado

Se debe importar las siguientes librerías.

```
import org.apache.spark.streaming.Seconds
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.dstream.ReceiverInputDStream
```

Instanciamos un objeto `StreamingContext` usando un `SparkContext` existente con un intervalo de 20 segundos.

```
val streamingContext: StreamingContext = new StreamingContext(sc, Seconds(20))
```

Del objeto `streamingContext` la función `socketTextStream` recibe dos parámetros: datos de un socket en el host y el puerto, para luego crear una conexión TCP.

```
val lines: ReceiverInputDStream[String] = streamingContext.socketTextStream("localhost",
  ↳ 9999)
```

## 2.3. RDD

```
val cadenas = Array("Docentes", "inteligenciaArtificial", "quefinal")
val cadenasRDD = sc.parallelize(cadenas)
cadenasRDD.collect()
file.collect()
val filtro = cadenasRDD.filter(line => line.contains("quefinal"))
val fileNotFound = sc.textFile("/7afljdljsjd/alkls/", 6)
fileNotFound.collect()
val cadenas = Array("Docentes", "inteligenciaArtificial",
  ↳ "quefinal")
```

### 2.3.1. Código Refactorizado

Se crea una `Array` de cadenas llamada `cadenas`].

```
val cadenas = Array("Docentes", "inteligenciaArtificial", "quefinal")
```

La función o método `parallelize` recibe como parámetro un array, en este caso el array de `cadenas` y crea un RDD de cadenas utilizando la variable `sc` que nos provee `SparkContext`.

```
val cadenasRDD = sc.parallelize(cadenas)
```

El método `collect`, recupera todos los elementos del RDD.

```
cadenasRDD.collect()
```



Con el metodo **filter** retornamos todas las cadenas bajo la condicion de un predicado, en este caso todas las cadenas que contienen **quefinal**

```
val filtro = cadenasRDD.filter(line => line.contains("quefinal"))
```

El metodo `textFile` lee un archivo de texto de HDFS y devuelve como un RDD de cadenas. En el caso solo intenta leer el archivo ya que no existe. La funcion esta parametrizada donde primero se le envia la ruta del archivo `/7añljdsjd/alkls/` y `6` que es el numero minimo de particiones para Hadoop RDD.

```
val fileNotFound = sc.textFile("/7añljdsjd/alkls/", 6)
```

Ya no se puede ejecutar la siguiente linea a menos de que exista un archivo en la linea anterior, nos saldrá un excepción de **FileNotFoundException**.

```
fileNotFound.collect()
```