**Apache Spark Installation on Windows**

**https://sparkbyexamples.com/spark/apache-spark-installation-on-windows/**

* Post category:[Apache Spark](https://sparkbyexamples.com/category/spark/)
* Post last modified:January 27, 2023

**Spread the love**

In this article, I will explain step-by-step how to do Apache Spark Installation on windows os 7, 10, and the latest version and also explain how to [start a history server](https://sparkbyexamples.com/spark/spark-history-server-to-monitor-applications/) and [monitor your jobs using Web UI](https://sparkbyexamples.com/spark/spark-web-ui-understanding/).

**Related:**

* [Spark Install Latest Version on Mac](https://sparkbyexamples.com/spark/install-apache-spark-on-mac/)
* [PySpark Install on Windows](https://sparkbyexamples.com/pyspark/how-to-install-and-run-pyspark-on-windows/)

**Install Java 8 or Later**

To install Apache Spark on windows, you would need Java 8 or the latest version hence download the Java version from [Oracle](https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html) and install it on your system. If you wanted OpenJDK you can download it from [here](https://adoptopenjdk.net/).

After download, double click on the downloaded .exe (jdk-8u201-windows-x64.exe) file in order to install it on your windows system. Choose any custom directory or keep the default location.

**Note:** This article explains Installing Apache Spark on Java 8, same steps will also work for Java 11 and 13 versions.

**Apache Spark Installation on Windows**

Apache Spark comes in a compressed tar/zip files hence installation on windows is not much of a deal as you just need to download and untar the file. Download Apache spark by accessing the [Spark Download page](https://spark.apache.org/downloads.html) and select the link from “Download Spark (point 3 from below screenshot)”.

If you wanted to use a different version of Spark & Hadoop, select the one you wanted from the drop-down; the link on point 3 changes to the selected version and provides you with an updated link to download.



After download, untar the binary using [7zip](https://www.7-zip.org/download.html) or any zip utility to extract the zip file and copy the extracted directory spark-3.0.0-bin-hadoop2.7 to c:\apps\opt\spark-3.0.0-bin-hadoop2.7

**Spark Environment Variables**

Post Java and Apache Spark installation on windows, set JAVA\_HOME, SPARK\_HOME, HADOOP\_HOME and PATH environment variables. If you know how to set the environment variable on windows, add the following.

JAVA\_HOME = C:\Program Files\Java\jdk1.8.0\_201

PATH = %PATH%;%JAVA\_HOME%

SPARK\_HOME = C:\apps\opt\spark-3.0.0-bin-hadoop2.7

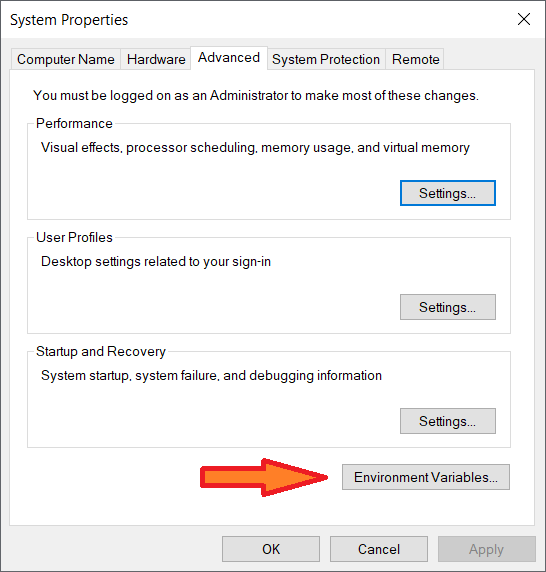
HADOOP\_HOME = C:\apps\opt\spark-3.0.0-bin-hadoop2.7

PATH=%PATH%;%SPARK\_HOME%

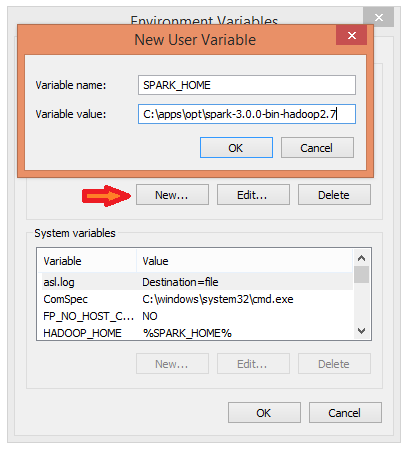
Copy

Follow the below steps if you are not aware of how to add or edit environment variables on windows.

1. Open System Environment Variables window and select **Environment Variables**.

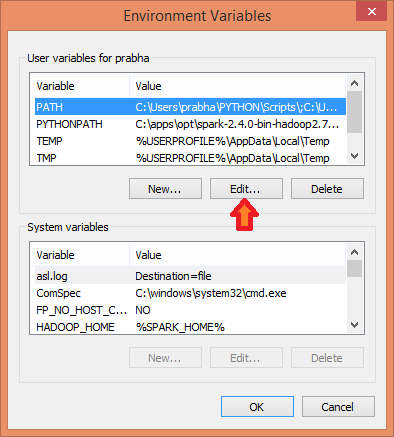


2. On the following Environment variable screen, add SPARK\_HOME, HADOOP\_HOME, JAVA\_HOME by selecting the **New** option.

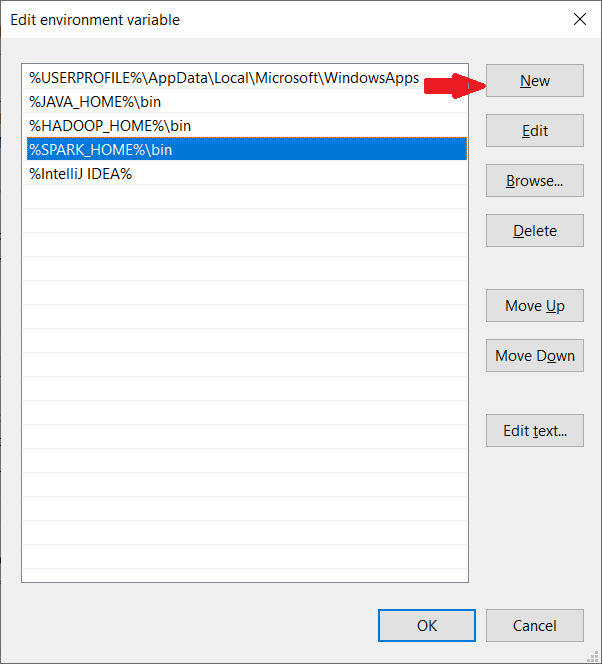


3. This opens up the **New User Variables** window where you can enter the variable name and value.

4. Now **Edit** the **PATH** variable



5. Add Spark, Java, and Hadoop bin location by selecting **New** option.



**Spark with winutils.exe on Windows**

Many beginners think Apache Spark needs a Hadoop cluster installed to run but that’s not true, Spark can run on AWS by using S3, Azure by using blob storage without Hadoop and HDFSe.t.c.

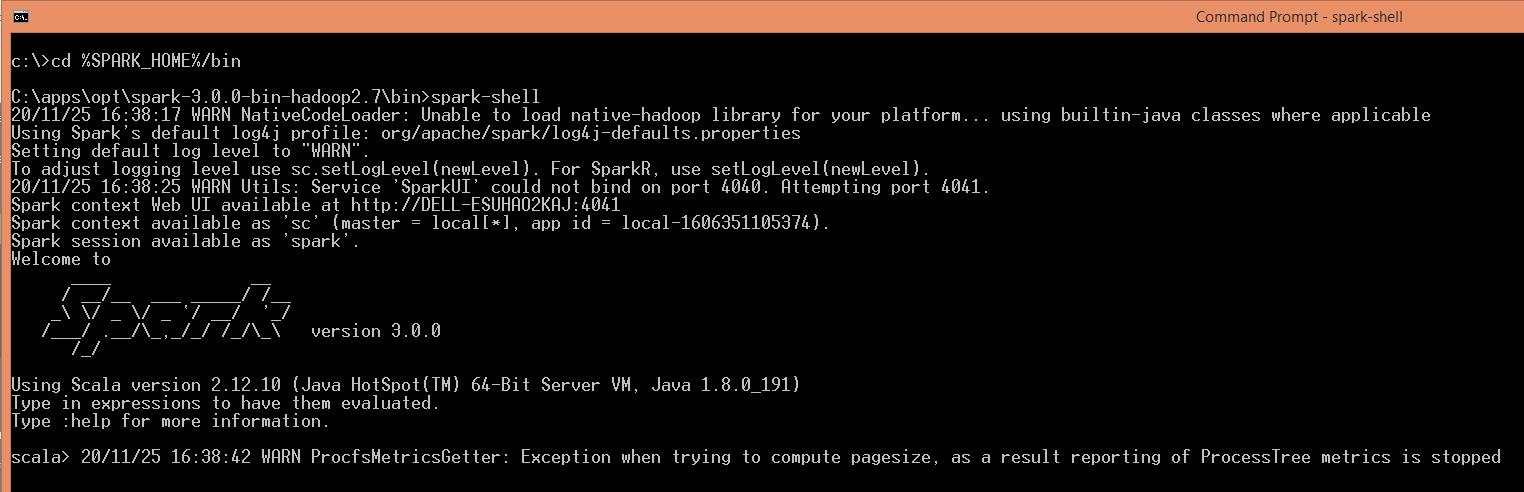
To run Apache Spark on windows, you need winutils.exe as it uses POSIX like file access operations in windows using windows API.

winutils.exe enables Spark to use Windows-specific services including running shell commands on a windows environment.

[Download winutils.exe for Hadoop 2.7](https://github.com/steveloughran/winutils/blob/master/hadoop-2.7.1/bin/winutils.exe) and copy it to %SPARK\_HOME%\bin folder. Winutils are different for each Hadoop version hence download the right version based on your Spark vs Hadoop distribution from <https://github.com/steveloughran/winutils>

**Apache Spark shell**

spark-shell is a CLI utility that comes with Apache Spark distribution, open command prompt, go to cd %SPARK\_HOME%/bin and type spark-shell command to run Apache Spark shell. You should see something like below (ignore the error you see at the end). Sometimes it may take a minute or two for your Spark instance to initialize to get to the below screen.

Spark Shell Command Line

Spark-shell also creates a [Spark context web UI](https://sparkbyexamples.com/spark/spark-web-ui-understanding/) and by default, it can access from [http://localhost:4040](http://localhost:4040/). —-- In some systems use 4041

On spark-shell command line, you can run any Spark statements like creating an RDD, getting Spark version e.t.c

scala> spark.version

res2: String = 3.0.0

scala> val rdd = sc.parallelize(Array(1,2,3,4,5,6,7,8,9,10))

rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at console:24

scala>

Copy

This completes the installation of Apache Spark on Windows 7, 10, and any latest.

**Quit the spark-shell prompt**

**ctrl+d**

**or**

**:quit**

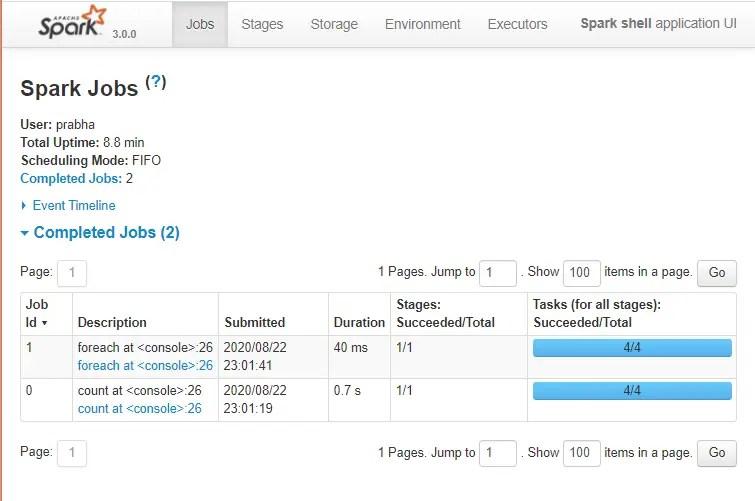
**Where to go Next?**

You can continue following the below document to see how you can debug the logs using Spark Web UI and enable the Spark history server or follow the links as next steps

* [Spark RDD Tutorial](https://sparkbyexamples.com/spark-rdd-tutorial/)
* [Spark Hello World Example in IntelliJ IDEA](https://sparkbyexamples.com/spark/spark-hello-world-example-in-intellij/)

**Web UI on Windows**

Apache Spark provides a suite of Web UIs (Jobs, Stages, Tasks, Storage, Environment, Executors, and SQL) to [monitor the status of your Spark application](https://sparkbyexamples.com/spark/spark-web-ui-understanding/), resource consumption of Spark cluster, and Spark configurations. On Spark Web UI, you can see how the operations are executed.

Spark Web UI

**History Server**

History server keeps a log of all Spark applications you submit by spark-submit, spark-shell. You can enable Spark to collect the logs by adding the below configs to spark-defaults.conf file, conf file is located at %SPARK\_HOME%/conf directory.

spark.eventLog.enabled true

spark.history.fs.logDirectory file:///c:/logs/path

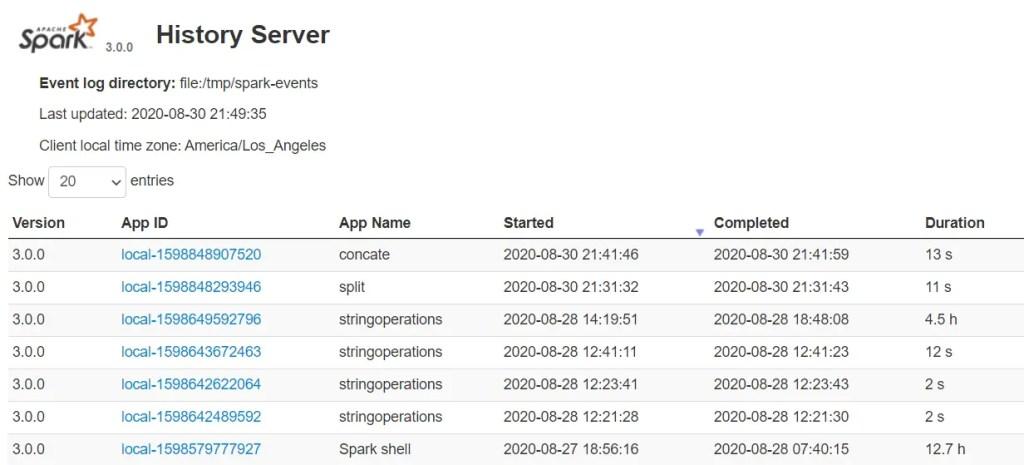
Copy

After setting the above properties, start the history server by starting the below command.

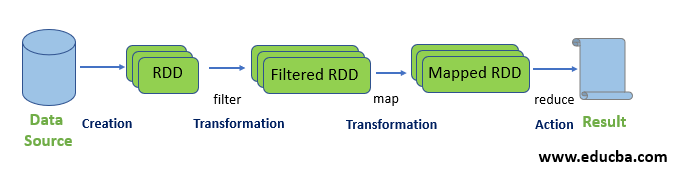
$SPARK\_HOME/bin/spark-class.cmd org.apache.spark.deploy.history.HistoryServer

Copy

By default History server listens at 18080 port and you can access it from browser using <http://localhost:18080/>

Spark History Server

By clicking on each App ID, you will get the details of the application in Spark web UI.

****

# Interactive Analysis with the Spark Shell

<https://spark.apache.org/docs/latest/quick-start.html>

<https://www.educba.com/spark-commands/>

<https://data-flair.training/blogs/scala-spark-shell-commands/>

<https://intellipaat.com/blog/tutorial/spark-tutorial/loading-and-saving-your-data/#:~:text=Saving%20the%20text%20files%3A%20Spark,write%20output%20from%20multiple%20codes>.

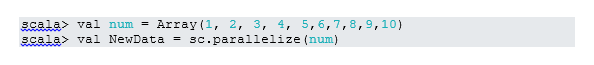
<https://www.educba.com/spark-commands/>

Load the files

scala> val input = sc.textFile("D:/test.txt")

input: org.apache.spark.rdd.RDD[String] = D:/test.txt MapPartitionsRDD[2] at textFile at <console>:23

“sc” is the spark context

**Create RDD through parallelizing**NewData is the RDD now.

**Count Items in RDD**

**RDD 2**

**Read the first 3 Items from RDD**

**scala> newd.take(4)**

**res1: Array[Int] = Array(1, 2, 3, 4)**

**Collect**

**This function returns all RDD’s content to the driver program. This helps debug at various steps of the writing program.**

**scala> newd.collect()**

**res2: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7)**

**scala> newd**

**res3: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:24**

**scala> print(newd)**

**ParallelCollectionRDD[0] at parallelize at <console>:24**

### Show Contents From Spark (Scala)

****

**val dept = List(("Finance",10),("Marketing",20),**

**("Sales",30), ("IT",40))**

**val rdd=spark.sparkContext.parallelize(dept)**

**val dataColl=rdd.collect()**

**dataColl.foreach(println)**

**Copy**

**This displays the contents of an RDD as a tuple to the console.**

****

**(Finance,10)**

**(Marketing,20)**

**(Sales,30)**

**(IT,40)**

**Copy**

**If you wanted to retrieve the individual elements do the following.**

****

**dataColl.foreach(f=>println(f.\_1 +","+f.\_2))**

**Copy**

**This yields the below output.**

****

**Finance,10**

**Marketing,20**

**Sales,30**

**IT,40**

****

**val dataCollLis=rdd.collectAsMap()**

**dataCollLis.foreach(f=>println(f.\_1 +","+f.\_2))**

**IT,40**

**Marketing,20**

**Sales,30**

**Finance,10**

**Save output/processed data into the text file**

**RDD 4**

**Create a file in notepad as “f1.scala”**

**val dept = List(("Finance",10),("Marketing",20),**

**("Sales",30), ("IT",40))**

**val rdd=spark.sparkContext.parallelize(dept)**

**val dataColl=rdd.collect()**

**dataColl.foreach(println)**

**Run the program in spark-shell as**

**scala> :load D:\f1.scala**

**Loading D:\f1.scala...**

**dept: List[(String, Int)] = List((Finance,10), (Marketing,20), (Sales,30), (IT,40))**

**rdd: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[2] at parallelize at D:\f1.scala:23**

**dataColl: Array[(String, Int)] = Array((Finance,10), (Marketing,20), (Sales,30), (IT,40))**

**(Finance,10)**

**(Marketing,20)**

**(Sales,30)**

**(IT,40)**

**Run the program in command line**

**c:\> spark-shell -i file.scala**

****

<https://sparkbyexamples.com/spark/spark-read-text-file-rdd-dataframe/>

### textFile() – Read text file into RDD

sparkContext.textFile() method is used to read a text file from HDFS, S3 and any Hadoop supported file system, this method takes the path as an argument and optionally takes a number of partitions as the second argument.

println("##spark read text files from a directory into RDD")

val rddFromFile = spark.sparkContext.textFile("src/main/resources/csv/text01.txt")

println(rddFromFile.getClass)

println("##Get data Using collect")

rddFromFile.collect().foreach(f=>{

println(f)

})

OUTPUT:

##spark read text files from a directory into RDD

class org.apache.spark.rdd.MapPartitionsRDD

##Get data Using collect

One,1

Eleven,11

—-----------------------------------------------------

scala> :help

All commands can be abbreviated, e.g., :he instead of :help.

:completions <string> output completions for the given string

:edit <id>|<line> edit history

:help [command] print this summary or command-specific help

:history [num] show the history (optional num is commands to show)

:h? <string> search the history

:imports [name name ...] show import history, identifying sources of names

:implicits [-v] show the implicits in scope

:javap <path|class> disassemble a file or class name

:line <id>|<line> place line(s) at the end of history

:load <path> interpret lines in a file

:paste [-raw] [path] enter paste mode or paste a file

:power enable power user mode

:quit exit the interpreter

:replay [options] reset the repl and replay all previous commands

:require <path> add a jar to the classpath

:reset [options] reset the repl to its initial state, forgetting all session entries

:save <path> save replayable session to a file

:sh <command line> run a shell command (result is implicitly => List[String])

:settings <options> update compiler options, if possible; see reset

:silent disable/enable automatic printing of results

:type [-v] <expr> display the type of an expression without evaluating it

:kind [-v] <type> display the kind of a type. see also :help kind

:warnings show the suppressed warnings from the most recent line which had any

**https://sparkbyexamples.com/spark/spark-read-text-file-rdd-dataframe/**

**LOAD DATA FROM CSV**

create a data file in D:\data.csv

Arun, 133

Kumar, 564

Kala, 565

Balu, 354

Mala, 343

create a program in D:\display.scala

println("##spark read text files from a directory into RDD")

val rddFromFile = spark.sparkContext.textFile("d:/data.csv")

println(rddFromFile.getClass)

println("##Get data Using collect")

rddFromFile.collect().foreach(f=>{

println(f)

})

Run the program in spark-shell as

scala> :load d:\display.scala

Loading d:\f2.scala...

##spark read text files from a directory into RDD

rddFromFile: org.apache.spark.rdd.RDD[String] = d:/data.csv MapPartitionsRDD[1] at textFile at d:\f2.scala:22

class org.apache.spark.rdd.MapPartitionsRDD

##Get data Using collect

Arun, 133

Kumar, 564

Kala, 565

Balu, 354

Mala, 343

### wholeTextFiles() – Read text files into RDD of Tuple.

sparkContext.wholeTextFiles() reads a text file into PairedRDD of type RDD[(String,String)] with the key being the file path and value being contents of the file.

println("##read whole text files")

val rddWhole = spark.sparkContext.wholeTextFiles("d:/data.csv")

println(rddWhole.getClass)

rddWhole.foreach(f=>{

println(f.\_1+"=>"+f.\_2)

})

scala> :load d:\2wholeKV.scala

Loading d:\2wholeKV.scala...

##read whole text files

rddWhole: org.apache.spark.rdd.RDD[(String, String)] = d:/data.csv MapPartitionsRDD[5] at wholeTextFiles at d:\2wholeKV.scala:22

class org.apache.spark.rdd.MapPartitionsRDD

file:/d:/data.csv=>Arun, 133

Kumar, 564

Kala, 565

Balu, 354

Mala, 343

## text() – Read text file into DataFrame

spark.read.text() method is used to read a text file into DataFrame. like in RDD, we can also use this method to read multiple files at a time, reading patterns matching files and finally reading all files from a directory.

//returns DataFrame

val df = spark.read.text("d:/data.csv")

df.printSchema()

df.show(false)

scala> :load d:/3dataframe.scala

Loading d:\3dataframe.scala...

df: org.apache.spark.sql.DataFrame = [value: string]

root

|-- value: string (nullable = true)

+----------+

|value |

+----------+

|Arun, 133 |

|Kumar, 564|

|Kala, 565 |

|Balu, 354 |

|Mala, 343 |

+----------+

Dataframe with columns

val df = spark.read.text("d:/data.csv")

//converting to columns by splitting

import spark.implicits.\_

val df2 = df.map(f=>{

val elements = f.getString(0).split(",")

(elements(0),elements(1))

})

df2.printSchema()

df2.show(false)

scala> :load d:/4dfcol.scala

Loading d:\4dfcol.scala...

df: org.apache.spark.sql.DataFrame = [value: string]

import spark.implicits.\_

df2: org.apache.spark.sql.Dataset[(String, String)] = [\_1: string, \_2: string]

root

|-- \_1: string (nullable = true)

|-- \_2: string (nullable = true)

+-----+----+

|\_1 |\_2 |

+-----+----+

|Arun | 133|

|Kumar| 564|

|Kala | 565|

|Balu | 354|

|Mala | 343|

+-----+----+

## textFile() – Read text file into Dataset

// returns Dataset[String]

val ds = spark.read.textFile("d:/data.csv")

ds.printSchema()

ds.show(false)

//converting to columns by splitting

import spark.implicits.\_

val ds2 = ds.map(f=> {

val elements = f.split(",")

(elements(0),elements(1))

})

ds2.printSchema()

ds2.show(false)

scala> :load d:/5ds.scala

Loading d:\5ds.scala...

ds: org.apache.spark.sql.Dataset[String] = [value: string]

root

|-- value: string (nullable = true)

+----------+

|value |

+----------+

|Arun, 133 |

|Kumar, 564|

|Kala, 565 |

|Balu, 354 |

|Mala, 343 |

+----------+

import spark.implicits.\_

ds2: org.apache.spark.sql.Dataset[(String, String)] = [\_1: string, \_2: string]

root

|-- \_1: string (nullable = true)

|-- \_2: string (nullable = true)

+-----+----+

|\_1 |\_2 |

+-----+----+

|Arun | 133|

|Kumar| 564|

|Kala | 565|

|Balu | 354|

|Mala | 343|

+-----+----+

## Spark Core

In this section of the Apache Spark Tutorial, you will learn different concepts of the Spark Core library with examples in Scala code. Spark Core is the main base library of the Spark which provides the abstraction of how distributed task dispatching, scheduling, basic I/O functionalities and etc.

Before getting your hands dirty on Spark programming, have your [Development Environment Setup to run Spark Examples using IntelliJ IDEA](https://sparkbyexamples.com/spark/spark-setup-run-intellij-on-windows/)

### SparkSession

[SparkSession](https://sparkbyexamples.com/spark/sparksession-explained-with-examples/) introduced in version 2.0, It is an entry point to underlying Spark functionality in order to programmatically use Spark RDD, DataFrame and Dataset. It’s object *spark* is default available in spark-shell.

Creating a SparkSession instance would be the first statement you would write to program with [RDD](https://sparkbyexamples.com/spark-rdd-tutorial/), [DataFrame](https://sparkbyexamples.com/spark/different-ways-to-create-a-spark-dataframe/) and Dataset. SparkSession will be created using SparkSession.builder() builder pattern.



import org.apache.spark.sql.SparkSession

val spark:SparkSession = SparkSession.builder()

.master("local[1]")

.appName("SparkByExamples.com")

.getOrCreate()

Copy

### Spark Context

[SparkContext](https://sparkbyexamples.com/spark/spark-sparkcontext/) is available since Spark 1.x (JavaSparkContext for Java) and is used to be an entry point to Spark and [PySpark](https://sparkbyexamples.com/pyspark-tutorial/) before introducing SparkSession in 2.0. Creating SparkContext was the first step to the program with RDD and to connect to Spark Cluster. It’s object sc by default available in spark-shell.

Since Spark 2.x version, When you create SparkSession, SparkContext object is by default create and it can be accessed using spark.sparkContext

Note that you can create just one SparkContext per JVM but can create many SparkSession objects.

## RDD Spark Tutorial

RDD ([Resilient Distributed Dataset)](https://sparkbyexamples.com/spark-rdd-tutorial/) is a fundamental data structure of Spark and it is the primary data abstraction in Apache Spark and the Spark Core. RDDs are fault-tolerant, immutable distributed collections of objects, which means once you create an RDD you cannot change it. Each dataset in RDD is divided into logical partitions, which can be computed on different nodes of the cluster.

This Apache Spark RDD Tutorial will help you start understanding and using Apache Spark RDD (Resilient Distributed Dataset) with Scala code examples. All RDD examples provided in this tutorial were also tested in our development environment and are available at [GitHub spark scala examples project](https://github.com/spark-examples/spark-scala-examples/tree/master/src/main/scala/com/sparkbyexamples/spark/rdd) for quick reference.

In this section of the Apache Spark tutorial, I will introduce the RDD and explains how to create them and use its transformation and action operations. Here is the [full article on Spark RDD](https://sparkbyexamples.com/spark-rdd-tutorial/) in case if you wanted to learn more of and get your fundamentals strong.

### RDD creation

RDD’s are created primarily in two different ways, first parallelizing an existing collection and secondly referencing a dataset in an external storage system (HDFS, HDFS, S3 and many more).

#### sparkContext.parallelize()

sparkContext.parallelize is used to parallelize an existing collection in your driver program. This is a basic method to create RDD.



//Create RDD from parallelize

val dataSeq = Seq(("Java", 20000), ("Python", 100000), ("Scala", 3000))

val rdd=spark.sparkContext.parallelize(dataSeq)

Copy

#### sparkContext.textFile()

Using textFile() method we can read a text (.txt) file from many sources like HDFS, S#, Azure, local e.t.c into RDD.



//Create RDD from external Data source

val rdd2 = spark.sparkContext.textFile("/path/textFile.txt")

Copy

### RDD Operations

On Spark RDD, you can perform two kinds of operations.

#### RDD Transformations

[Spark RDD Transformations](https://sparkbyexamples.com/apache-spark-rdd/spark-rdd-transformations/) are lazy operations meaning they don’t execute until you call an action on RDD. Since RDD’s are immutable, When you run a transformation(for example map()), instead of updating a current RDD, it returns a new RDD.

Some transformations on RDD’s are flatMap(), map(), reduceByKey(), filter(), sortByKey() and all these return a new RDD instead of updating the current.

#### RDD Actions

[RDD Action operation](https://sparkbyexamples.com/apache-spark-rdd/spark-rdd-actions/) returns the values from an RDD to a driver node. In other words, any RDD function that returns non RDD[T] is considered as an action. RDD operations trigger the computation and return RDD in a List to the driver program.

Some actions on RDD’s are count(), collect(), first(), max(), reduce() and more.

### RDD Examples

* [How to read multiple text files into RDD](https://sparkbyexamples.com/apache-spark-rdd/spark-read-multiple-text-files-into-a-single-rdd/)
* [Read CSV file into RDD](https://sparkbyexamples.com/apache-spark-rdd/spark-load-csv-file-into-rdd/)
* [Ways to create an RDD](https://sparkbyexamples.com/apache-spark-rdd/different-ways-to-create-spark-rdd/)
* [Create an empty RDD](https://sparkbyexamples.com/apache-spark-rdd/spark-how-to-create-an-empty-rdd/)
* [RDD Pair Functions](https://sparkbyexamples.com/apache-spark-rdd/spark-pair-rdd-functions/)
* [Generate DataFrame from RDD](https://sparkbyexamples.com/apache-spark-rdd/convert-spark-rdd-to-dataframe-dataset/)

## DataFrame Spark Tutorial with Basic Examples

DataFrame definition is very well explained by Databricks hence I do not want to define it again and confuse you. Below is the definition I took it from Databricks.

*DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as structured data files, tables in Hive, external databases, or existing RDDs.*

– Databricks

### DataFrame creation

The simplest way to create a DataFrame is from a seq collection. DataFrame can also be created from an RDD and by reading files from several sources.

#### using createDataFrame()

By using createDataFrame() function of the SparkSession you can create a DataFrame.



val data = Seq(('James','','Smith','1991-04-01','M',3000),

('Michael','Rose','','2000-05-19','M',4000),

('Robert','','Williams','1978-09-05','M',4000),

('Maria','Anne','Jones','1967-12-01','F',4000),

('Jen','Mary','Brown','1980-02-17','F',-1)

)

val columns = Seq("firstname","middlename","lastname","dob","gender","salary")

df = spark.createDataFrame(data), schema = columns).toDF(columns:\_\*)

Copy

Since DataFrame’s are structure format which contains names and column, we can get the schema of the DataFrame using df.printSchema()

df.show() shows the 20 elements from the DataFrame.



+---------+----------+--------+----------+------+------+

|firstname|middlename|lastname|dob |gender|salary|

+---------+----------+--------+----------+------+------+

|James | |Smith |1991-04-01|M |3000 |

|Michael |Rose | |2000-05-19|M |4000 |

|Robert | |Williams|1978-09-05|M |4000 |

|Maria |Anne |Jones |1967-12-01|F |4000 |

|Jen |Mary |Brown |1980-02-17|F |-1 |

+---------+----------+--------+----------+------+------+

Copy

In this Apache Spark SQL DataFrame Tutorial, I have explained several mostly used operation/functions on DataFrame & DataSet with working scala examples. This is a work in progress section where you will see more articles and samples are coming.

* [Different ways to create a DataFrame](https://sparkbyexamples.com/spark/different-ways-to-create-a-spark-dataframe/)
* [How to create an empty DataFrame](https://sparkbyexamples.com/spark/spark-how-to-create-an-empty-dataframe/)
* [How to create an empty DataSet](https://sparkbyexamples.com/spark/spark-how-to-create-an-empty-dataset/)
* [Spark DataFrame – Rename nested column](https://sparkbyexamples.com/spark/rename-a-column-on-spark-dataframes/)
* [How to add or update a column on DataFrame](https://sparkbyexamples.com/spark/spark-dataframe-withcolumn/)
* [How to drop a column on DataFrame](https://sparkbyexamples.com/spark/spark-drop-column-from-dataframe-dataset/)
* [Spark when otherwise usage](https://sparkbyexamples.com/spark/spark-case-when-otherwise-example/)
* [How to add literal constant to DataFrame](https://sparkbyexamples.com/spark/using-lit-and-typedlit-to-add-a-literal-or-constant-to-spark-dataframe/)
* [Spark Data Types explained](https://sparkbyexamples.com/spark/spark-sql-dataframe-data-types/)
* [How to change column data type](https://sparkbyexamples.com/spark/spark-change-dataframe-column-type/)
* [How to Pivot and Unpivot a DataFrame](https://sparkbyexamples.com/spark/how-to-pivot-table-and-unpivot-a-spark-dataframe/)
* [Create a DataFrame using StructType & StructField schema](https://sparkbyexamples.com/spark/spark-sql-structtype-on-dataframe/)
* [How to select the first row of each group](https://sparkbyexamples.com/spark/spark-dataframe-how-to-select-the-first-row-of-each-group/)
* [How to sort DataFrame](https://sparkbyexamples.com/spark/spark-how-to-sort-dataframe-column-explained/)
* [How to union DataFrame](https://sparkbyexamples.com/spark/spark-dataframe-union-and-union-all/)
* [How to drop Rows with null values from DataFrame](https://sparkbyexamples.com/spark/spark-dataframe-drop-rows-with-null-values/)
* [How to split single to multiple columns](https://sparkbyexamples.com/spark/spark-split-dataframe-column-into-multiple-columns/)
* [How to concatenate multiple columns](https://sparkbyexamples.com/spark/spark-concatenate-dataframe-columns/)
* [How to replace null values in DataFrame](https://sparkbyexamples.com/spark/spark-how-to-replace-null-values/)
* [How to remove duplicate rows on DataFrame](https://sparkbyexamples.com/spark/spark-sql-join-on-multiple-columns/)
* [How to remove distinct on multiple selected columns](https://sparkbyexamples.com/spark/spark-sql-distinct-multiple-columns/)
* [Spark map() vs mapPartitions()](https://sparkbyexamples.com/spark/spark-map-vs-mappartitions-transformation/)

[**https://sparkbyexamples.com/pyspark/how-to-install-and-run-pyspark-on-windows/**](https://sparkbyexamples.com/pyspark/how-to-install-and-run-pyspark-on-windows/)

* Spark Application UI: <http://localhost:4040/>
* Resource Manager: [http://localhost:9870](http://localhost:9870/)
* Spark JobTracker: <http://localhost:8088/>
* Node Specific Info: <http://localhost:8042/>

Note: To access these URLs, Spark application should in running state. If you wanted to access this URL regardless of your Spark application status and wanted to access Spark UI all the time, you would need to start [Spark History server](https://sparkbyexamples.com/hadoop/spark-setup-on-yarn/#spark-history-server).