BIG DATA

Table of Contents

Contents

[Introduction 4](#_Toc482581484)

[BIG DATA 4](#_Toc482581485)

[Hadoop 6](#_Toc482581486)

[Hadoop Architecture 6](#_Toc482581487)

[Advantages of Hadoop 7](#_Toc482581488)

[Hadoop Setup 8](#_Toc482581489)

[MapReduce 10](#_Toc482581490)

[The Algorithm 10](#_Toc482581491)

[Terminology 12](#_Toc482581492)

[Hadoop Distributed File System 13](#_Toc482581493)

[HDFS Architecture 13](#_Toc482581494)

[Namenode 14](#_Toc482581495)

[Datanode 14](#_Toc482581496)

[Block 15](#_Toc482581497)

[Starting HDFS 15](#_Toc482581498)

[Listing Files in HDFS 15](#_Toc482581499)

[Inserting Data into HDFS 15](#_Toc482581500)

[Retrieving Data from HDFS 16](#_Toc482581501)

[Shutting Down the HDFS 16](#_Toc482581502)

[YARN 17](#_Toc482581503)

[Components Of YARN 17](#_Toc482581504)

[HIVE 18](#_Toc482581505)

[Features 18](#_Toc482581506)

[Data Types 18](#_Toc482581507)

[Numeric Types 18](#_Toc482581508)

[Date/Time Types 18](#_Toc482581509)

[String Types 19](#_Toc482581510)

[Misc Types 19](#_Toc482581511)

[Complex Type 19](#_Toc482581512)

[Hive Commands 19](#_Toc482581513)

[Hive DDL Commands 19](#_Toc482581514)

[Create Database 19](#_Toc482581515)

[Drop database 20](#_Toc482581516)

[Creating Hive Tables 20](#_Toc482581517)

[Browse the table 20](#_Toc482581518)

[Altering and Dropping Tables 20](#_Toc482581519)

[Hive DML Commands 20](#_Toc482581520)

[LOAD DATA 21](#_Toc482581521)

[SELECTS and FILTERS 21](#_Toc482581522)

[GROUP BY 21](#_Toc482581523)

[Sort by 21](#_Toc482581524)

[Order by 21](#_Toc482581525)

[Inner joins 21](#_Toc482581526)

[Left outer joins 21](#_Toc482581527)

[Right outer joins 21](#_Toc482581528)

[Full outer joins 22](#_Toc482581529)

[Sqoop 23](#_Toc482581530)

[Sqoop Working 23](#_Toc482581531)

[Sqoop Installation 24](#_Toc482581532)

[Important Links 25](#_Toc482581533)

# Introduction

## BIG DATA

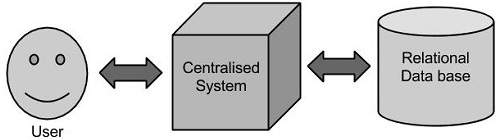
Data which are very large in size is called Big Data. Normally we work on data of size MB (WordDoc ,Excel) or maximum GB (Movies, Codes) but data in Peta bytes i.e. 10^15 byte size is called Big Data. It is stated that almost 90% of today's data has been generated in the past 3 years.

Thus Big Data includes huge **volume**, high **velocity**, and extensible **variety (Veractiy)** of data. These are called the 3V's of Big Data. The data in it will be of three types.

* **Structured data** : Relational data.
* **Semi Structured data** : XML data.
* **Unstructured data** : Word, PDF, Text, Any type of Logs.

**Traditional Approach**

In this approach, an enterprise will have a computer to store and process big data. Here data will be stored in an RDBMS like Oracle Database, MS SQL Server or DB2 and sophisticated software can be written to interact with the database, process the required data and present it to the users for analysis purpose.

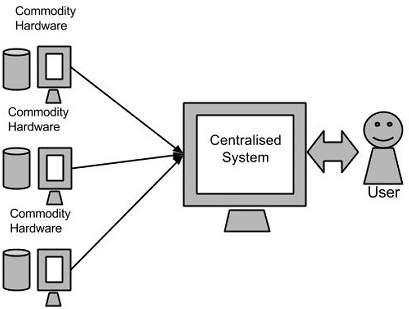


**Limitation**

This approach works well where we have less volume of data that can be accommodated by standard database servers, or up to the limit of the processor which is processing the data. But when it comes to dealing with huge amounts of data, it is really a tedious task to process such data through a traditional database server.

**Google’s Solution**

Google solved this problem using an ***algorithm called MapReduce.*** This algorithm divides the task into small parts and assigns those parts to many computers connected over the network, and collects the results to form the final result dataset.

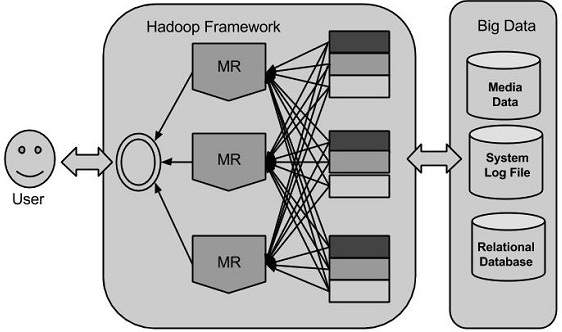


# Hadoop

Hadoop is an open source framework from Apache and is used to store process and analyze data which are very huge in volume. It is written in Java and is not OLAP (online analytical processing). Doug Cutting, Mike Cafarella and team took the solution provided by Google and started an Open Source Project called **HADOOP in 2005** and Doug named it after his son's toy elephant.

It is generally used for batch/offline processing. It is being used by Facebook, Yahoo, Google, Twitter, LinkedIn and many more. It can be easily scaled up just by adding nodes in the cluster. It runs applications using the MapReduce algorithm, where the data is processed in parallel on different CPU nodes.

Hadoop allows distributed processing of large datasets across clusters of computers using simple programming models. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.



## Hadoop Architecture

Hadoop framework includes following four modules:

* **Hadoop Common:** These are Java libraries and utilities required by other Hadoop modules. These libraries provides file system and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.
* **YARN:** This is a framework for job scheduling and cluster resource management.
* **Hadoop Distributed File System (HDFS):** A distributed file system that provides high-throughput access to application data.
* **MapReduce:** This is YARN-based system for parallel processing of large data sets using key value pairs.



## Advantages of Hadoop

* **Fast:** In HDFS the data distributed over the cluster and are mapped which helps in faster retrieval. Even the tools to process the data are often on the same servers, thus reducing the processing time. It is able to process terabytes of data in minutes and Peta bytes in hours.
* **Scalable:** Hadoop cluster can be extended by just adding nodes in the cluster.
* **Cost Effective:** Hadoop is open source and uses commodity hardware to store data so it really cost effective as compared to traditional relational database management system.
* **Resilient to failure:** HDFS has the property with which it can replicate data over the network, so if one node is down or some other network failure happens, then Hadoop takes the other copy of data and use it. Normally, data are replicated thrice but the replication factor is configurable.

## Hadoop Setup

1. Downloaded the hadoop gzip. Following link can be used for installation.

https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/SingleCluster.html#Installing\_Software

1. Copy gzip to machine and extract it.

tar xvf file.tar

1. Add following entry in .bashrc

export HADOOP\_HOME=/home/<user>/temp/hadoop-2.7.3

export PATH=$PATH:$HADOOP\_HOME/bin

1. Changes made in **hdfs-site.xml** for datanode and namenode directories

<property>

<name>dfs.permissions</name>

<value>false</value>

</property>

**<!-- Data node location -->**

<property>

<name>dfs.data.dir</name>

<value>/home/<user>/var/hadoop/dfs/name/data</value>

<final>true</final>

</property>

**<!-- Name node location -->**

<property>

<name>dfs.name.dir</name>

<value>/home/parag.garg/var/hadoop/dfs/name</value>

<final>true</final>

</property>

1. Create SSH connection between machines DEV05 and DEV06

for reference : http://www.tutorialspoint.com/hadoop/hadoop\_multi\_node\_cluster.htm

# MapReduce

Hadoop **MapReduce** is a framework for easily writing applications which process big amounts of data in-parallel on large multiple clusters. The term MapReduce actually refers to the following two different tasks that Hadoop programs perform:

* **The Map Task:** First task, which takes input data and converts it into a set of data, where individual elements are broken down into tuples (key/value pairs).
* **The Reduce Task:** This task takes the output from a map task as input and combines those data tuples into a smaller set of tuples. The reduce task is always performed after the map task.

MapReduce framework consists of a single master **JobTracker** and one slave **TaskTracker** per cluster-node. The master is responsible for resource management, tracking resource consumption/availability and scheduling the jobs component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves TaskTracker execute the tasks as directed by the master and provide task-status information to the master periodically.

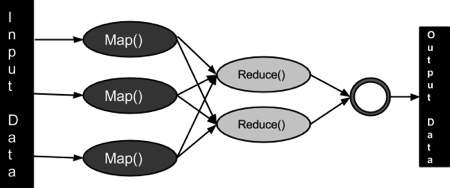
The JobTracker is a single point of failure for the Hadoop MapReduce service which means if JobTracker goes down, all running jobs are halted.

MapReduce is a processing technique and a program model for **distributed computing** based on java. The MapReduce algorithm contains **two important tasks, namely Map and Reduce.** Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to **scale data processing over multiple computing nodes**. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

## The Algorithm

* Generally MapReduce paradigm is based on sending the computer to where the data resides.
* MapReduce program executes in three stages, namely **map stage, shuffle stage, and reduce stage.**
  + **Map stage** : The map or mapper’s job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The **mapper processes the data and creates several small chunks of data.**
  + **Reduce stage** : This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer’s job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
* During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
* The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
* Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
* After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



The MapReduce framework operates on **<key, value> pairs**, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework. Input and Output types of a MapReduce job: (Input) <k1, v1> -> map -> <k2, v2>-> reduce -> <k3, v3>(Output).

|  |  |  |
| --- | --- | --- |
|  | **Input** | **Output** |
| **Map** | <k1, v1> | list (<k2, v2>) |
| **Reduce** | <k2, list(v2)> | list (<k3, v3>) |

## Terminology

* **PayLoad** - Applications implement the Map and the Reduce functions, and form the core of the job.
* **Mapper** - Mapper maps the input key/value pairs to a set of intermediate key/value pair.
* **NamedNode** - Node that manages the Hadoop Distributed File System (HDFS).
* **DataNode** - Node where data is presented in advance before any processing takes place.
* **MasterNode** - Node where JobTracker runs and which accepts job requests from clients.
* **SlaveNode** - Node where Map and Reduce program runs.
* **JobTracker** - Schedules jobs and tracks the assign jobs to Task tracker.
* **Task Tracker** - Tracks the task and reports status to JobTracker.
* **Job** - A program is an execution of a Mapper and Reducer across a dataset.
* **Task** - An execution of a Mapper or a Reducer on a slice of data.
* **Task Attempt** - A particular instance of an attempt to execute a task on a SlaveNode.

# Hadoop Distributed File System

Hadoop can work directly with any mountable distributed file system such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS).

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

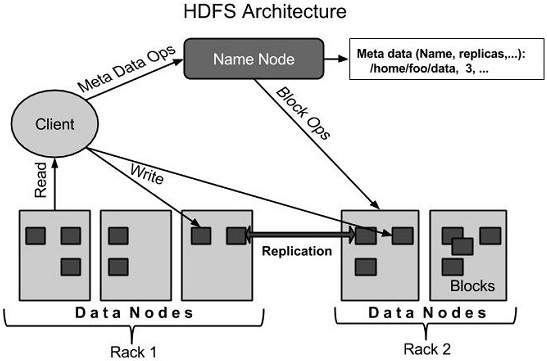
HDFS uses a master/slave architecture where master consists of a single **NameNode** that manages the file system metadata and one or more slave **DataNodes** that store the actual data.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode.

HDFS provides a shell like any other file system and a list of commands are available to interact with the file system. These shell commands will be covered in a separate chapter along with appropriate examples.

## HDFS Architecture

Given below is the architecture of a Hadoop File System.



HDFS follows the master-slave architecture and it has the following elements.

### Namenode

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks:

* Manages the file system namespace.
* Regulates client’s access to files.
* It also executes file system operations such as renaming, closing, and opening files and directories.

### Datanode

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

* Datanodes perform read-write operations on the file systems, as per client request.
* They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

### Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

## Starting HDFS

Initially you have to format the configured HDFS file system, open namenode (HDFS server), and execute the following command.

$ hadoop namenode -format

After formatting the HDFS, start the distributed file system. The following command will start the namenode as well as the data nodes as cluster.

$ start-dfs.sh

### Listing Files in HDFS

After loading the information in the server, we can find the list of files in a directory, status of a file, using ‘ls’. Given below is the syntax of ls that you can pass to a directory or a filename as an argument.

$ $HADOOP\_HOME/bin/hadoop fs -ls <args>

### Inserting Data into HDFS

Assume we have data in the file called file.txt in the local system which is ought to be saved in the hdfs file system. Follow the steps given below to insert the required file in the Hadoop file system.

**Step 1**

You have to create an input directory.

$ $HADOOP\_HOME/bin/hadoop fs -mkdir /user/input

**Step 2**

Transfer and store a data file from local systems to the Hadoop file system using the put command.

$ $HADOOP\_HOME/bin/hadoop fs -put /home/file.txt /user/input

**Step 3**

You can verify the file using ls command.

$ $HADOOP\_HOME/bin/hadoop fs -ls /user/input

### Retrieving Data from HDFS

Assume we have a file in HDFS called outfile. Given below is a simple demonstration for retrieving the required file from the Hadoop file system.

**Step 1**

Initially, view the data from HDFS using cat command.

$ $HADOOP\_HOME/bin/hadoop fs -cat /user/output/outfile

**Step 2**

Get the file from HDFS to the local file system using get command.

$ $HADOOP\_HOME/bin/hadoop fs -get /user/output/ /home/hadoop\_tp/

## Shutting Down the HDFS

You can shut down the HDFS by using the following command.

$ stop-dfs.sh

# YARN

Yet Another Resource Manager takes programming to the next level beyond Java , and makes it interactive to let another application Hbase, Spark etc. to work on it. Different Yarn applications can co-exist on the same cluster so MapReduce, Hbase, Spark all can run at the same time bringing great benefits for manageability and cluster utilization.

## Components Of YARN

* **Client:** For submitting MapReduce jobs.
* **Resource Manager:**To manage the use of resources across the cluster
* **Node Manager:** For launching and monitoring the computer containers on machines in the cluster.
* **Map Reduce Application Master:**Checks tasks running the MapReduce job. The application master and the MapReduce tasks run in containers that are scheduled by the resource manager, and managed by the node managers.

Jobtracker & Tasktracker were used in previous version of Hadoop, which were responsible for handling resources and checking progress management. However, Hadoop 2.0 has Resource manager and Node Manager to overcome the shortfall of Jobtracker & Tasktracker.

# HIVE

Apache Hive is a data ware house system for Hadoop that runs **SQL like queries called HQL (Hive query language)** which gets internally converted to map reduce jobs. Hive was developed by Facebook. It supports Data definition Language, Data Manipulation Language and user defined functions.

**DDL:** create table, create index, create views.

**DML:** Select, Where, group by, Join, Order By

## Features

* Hive is not RDBMS.
* Hive is not for OLTP (Online Transaction Processing).
* Hive's metastore is used to persist schema i.e. table definition (table name, columns, types), location of table files, row format of table files, storage format of files.
* Hive Query Language is similar to SQL and gets reduced to map reduce jobs in backend.
* Hive's default database is **Apache** **derby**.

## Data Types

Hive data types are categorized in numeric types, string types, misc types and complex types. A list of Hive data types are given below.

### Numeric Types

**TINYINT** (1-byte signed integer, from -128 to 127)

**SMALLINT** (2-byte signed integer, from -32,768 to 32,767)

**INT** (4-byte signed integer, from -2,147,483,648 to 2,147,483,647)

**BIGINT** (8-byte signed integer, from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)

**FLOAT** (4-byte single precision floating point number)

**DOUBLE** (8-byte double precision floating point number)

**DECIMAL** (Hive 0.13.0 introduced user definable precision and scale)

### Date/Time Types

TIMESTAMP  
DATE

### String Types

STRING  
VARCHAR  
CHAR

### Misc Types

BOOLEAN  
BINARY

### Complex Type

**arrays:** ARRAY<data\_type>

**maps:** MAP<primitive\_type, data\_type>

**structs:** STRUCT<col\_name : data\_type [COMMENT col\_comment], ...>

**union:** UNIONTYPE<data\_type, data\_type, ...>

## Hive Commands

Hive supports DDL, DML and User defined functions.

## Hive DDL Commands

* create database
* drop database
* create table
* drop table
* alter table
* create index
* create views

### Create Database

A database in Hive is a namespace or a collection of tables.

hive**>** CREATE SCHEMA userdb;

hive**>** SHOW DATABASES;

### Drop database

hive**>** DROP DATABASE IF EXISTS userdb;

### Creating Hive Tables

Create a table called Sonoo with two columns, the first being an integer and the other a string.

hive**>** CREATE TABLE Sonoo(foo INT, bar STRING);

Create a table called HIVE\_TABLE with two columns and a partition column called ds. The partition column is a virtual column. It is not part of the data itself but is derived from the partition that a particular dataset is loaded into.By default, tables are assumed to be of text input format and the delimiters are assumed to be ^A(ctrl-a).

hive> CREATE TABLE HIVE\_TABLE (foo INT, bar STRING) PARTITIONED BY (ds STRING);

### Browse the table

hive>  Show tables;

### Altering and Dropping Tables

hive**>** ALTER TABLE Sonoo RENAME TO Kafka;

hive**>** ALTER TABLE Kafka ADD COLUMNS (col INT);

hive**>** ALTER TABLE HIVE\_TABLE ADD COLUMNS (col1 INT COMMENT 'a comment');

hive**>** ALTER TABLE HIVE\_TABLE REPLACE COLUMNS (col2 INT, weight STRING, baz INT COMMENT 'baz replaces new\_col1');

## Hive DML Commands

* Select
* Where
* Group By
* Order By
* Load Data
* Join:
  + Inner Join
  + Left Outer Join
  + Right Outer Join
  + Full Outer Join

### LOAD DATA

hive**>** **LOAD DATA** LOCAL INPATH './usr/temp/file.txt'  **OVERWRITE** INTO

TABLE Employee;

### SELECTS and FILTERS

hive> **SELECT**  E.EMP\_ID FROM Employee E  **WHERE** E.Name = 'Gaurav';

### GROUP BY

hive> SELECT  E.EMP\_ID FROM Employee E **GROUP BY** E.Addresss;

### Sort by

hive**>** SELECT  E.EMP\_ID FROM Employee E **SORT BY** E.empid;

* May use multiple reducers for final output.
* Only guarantees ordering of rows within a reducer.
* May give partially ordered result.

### Order by

hive**>** SELECT  E.EMP\_ID FROM Employee E **order** BY E.empid;

* Uses single reducer to guarantee total order in output.
* LIMIT can be used to minimize sort time.

### Inner joins

Select  \*  from employee join employeedepartment  ON (employee.empid=employeedepartment.empId)

Output : <<InnerJoin.png>>

### Left outer joins

Select e.empId, empName, department from employee e **Left outer join** department ed on (e.empId=ed.empId);

Output : <<LeftOuterJoin.png>>

### Right outer joins

Select e.empId, empName, department from employee e **Right outer join** department ed on (e.empId=ed.empId);

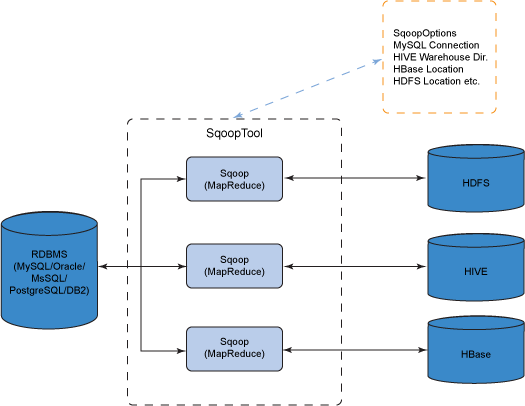
### Full outer joins

Select e.empId, empName, department from employee e **FULL outer join** department ed on (e.empId=ed.empId);

# Sqoop

Sqoop is an open source framework provided by Apache. It is a command-line interface application for transferring data between relational databases and Hadoop.

It supports incremental loads of a single table or a free form SQL query as well as saved jobs which can be run multiple times to import updates made to a database since the last import. Using Sqoop, data can be moved into HDFS/hive/hbase from MySQL/PostgreSQL/Oracle/SQL Server/DB2 and vice versa.



## Sqoop Working

When we request to import data to HDFS then following things happen.

**Step 1:** Sqoop send the request to Relational DB to send back metadata information about the table.

**Step 2:** From the received information it will generate the java classes. Therefore we should first configure Java, as Sqoop internally uses JDBC API to generate data.

**Step 3:** Now Sqoop tries to package the compiled classes to be able to generate table structure , post compiling creates jar file (Java packaging standard).

**Summary of steps**

* Sqoop request for metadata information from Relation DB.
* Relational DB returns the required request.
* Based on metadata information, Sqoop generates java classes.
* Based on primary id partitioning happens in table as multiple mappers will importing data as the same time

## Sqoop Installation

# Important Links

https://www.tutorialspoint.com/hadoop/

https://www.javatpoint.com/hadoop-tutorial

http://www.tutorialspoint.com/hadoop/hadoop\_multi\_node\_cluster.htm

https://letsdobigdata.wordpress.com/2014/01/13/setting-up-hadoop-1-2-1-multi-node-cluster-on-amazon-ec2-part-2/

https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/SingleCluster.html#Installing\_Software

http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html