***HDFS COMMANDS***

|  |  |  |
| --- | --- | --- |
| **Command** | **Windows Command Example** | **Mac M1 Equivalent Command Example** |

**Purpose**

|  |  |  |
| --- | --- | --- |
| **Start Hadoop services** | **sbin/start-all.cmd** | **sbin/start-all.sh or sbin/start-dfs.sh && sbin/start-yarn.sh** |
| **Check Hadoop services** | **jps** | **jps (same command)** |
| **List**  **files/directorie s** | **hdfs dfs -ls /path** | **hdfs dfs -ls /path** |
| **Create**  **directory** | **hdfs dfs -mkdir /path** | **hdfs dfs -mkdir /path** |
| **Create empty file** | **hdfs dfs -touchz /path** | **hdfs dfs -touchz /path** |
| **Copy from**  **local to HDFS** | **hdfs dfs -copyFromLocal localfile /path or hdfs dfs**  **-put** | **hdfs dfs -copyFromLocal localfile /path or hdfs dfs -put localfile**  **/path** |

|  |  |  |
| --- | --- | --- |
| **Show file contents** | **hdfs dfs -cat /path** | **hdfs dfs -cat /path** |
| **Copy from**  **HDFS to local** | **hdfs dfs -copyToLocal /path localdir or hdfs dfs -get** | **hdfs dfs -copyToLocal /path localdir or hdfs dfs -get /path**  **localdir** |
| **Show size of files** | **hdfs dfs -du /dirname** | **hdfs dfs -du /dirname** |
| **Show total size** | **hdfs dfs -dus /dirname** | **hdfs dfs -dus /dirname** |
| **Remove file** | **hdfs dfs -rm /path** | **hdfs dfs -rm /path** |
| **Remove**  **directory** | **hdfs dfs -rmdir /path** | **hdfs dfs -rmdir /path** |
| **Move files**  **within HDFS** | **hdfs dfs -mv /source /destination** | **hdfs dfs -mv /source /destination** |
| **Copy files**  **within HDFS** | **hdfs dfs -cp /source /destination** | **hdfs dfs -cp /source /destination** |
| **Append to file** | **hdfs dfs -appendToFile - /source/destination** | **hdfs dfs -appendToFile - /source/destination** |

|  |  |  |
| --- | --- | --- |
| **Count**  **files/directorie s** | **hdfs dfs -count /path** | **hdfs dfs -count /path** |
| **Show last part of file** | **hdfs dfs -tail /data/folder** | **hdfs dfs -tail /data/folder** |
| **Show first part of file** | **hdfs dfs -head /path** | **hdfs dfs -head /path** |
| **Test if path exists** | **hdfs dfs -test -e /path** | **hdfs dfs -test -e /path** |
| **Test if**  **directory** | **hdfs dfs -test -d /path** | **hdfs dfs -test -d /path** |
| **Test if file** | **hdfs dfs -test -f /path** | **hdfs dfs -test -f /path** |
| **Test if file empty** | **hdfs dfs -test -z /path** | **hdfs dfs -test -z /path** |
| **Move from**  **local to HDFS** | **hdfs dfs -moveFromLocal local\_source**  **/hdfs\_destination** | **hdfs dfs -moveFromLocal**  **local\_source /hdfs\_destination** |
| **Merge files** | **hdfs dfs -getmerge -nl /source /localdestination** | **hdfs dfs -getmerge -nl /source /localdestination** |

|  |  |  |
| --- | --- | --- |
| **Get checksum** | **hdfs dfs -checksum /file** | **hdfs dfs -checksum /file** |
| **Change group** | **hdfs dfs -chgrp**  **new\_group\_name /file** | **hdfs dfs -chgrp new\_group\_name /file** |
| **Show last**  **modified time** | **hdfs dfs -stat /path** | **hdfs dfs -stat /path** |
| **Empty trash** | **hdfs dfs -expunge** | **hdfs dfs -expunge** |
| **Change**  **owner/group** | **hdfs dfs -chown owner: group /path** | **hdfs dfs -chown owner: group /path** |
| **Change**  **permissions** | **hdfs dfs -chmod 777 /path** | **hdfs dfs -chmod 777 /path** |
| **Set replication factor** | **hdfs dfs -setrep -w n /path** | **hdfs dfs -setrep -w n /path** |

**●​ Basic HDFS Commands:**   
 **●​ jps: Lists running Hadoop Java processes.**

**●​ ls: Lists files and directories in HDFS.​**  
 **Syntax: hdfs dfs -ls /path**   
**●​ mkdir: Creates a directory in HDFS.​**  
 **Syntax: hdfs dfs -mkdir /path**   
**●​ touchz: Creates an empty file in HDFS.​**  
 **Syntax: hdfs dfs -touchz /path**

**●​ copyFromLocal / put: Copies files from local filesystem to HDFS.​** **Syntax: hdfs dfs -copyFromLocal /localpath /hdfspath or hdfs**  **dfs -put /localpath /hdfspath**   
**●​ cat: Displays file content.​**  
 **Syntax: hdfs dfs -cat /path**   
**●​ copyToLocal / get: Copies files from HDFS to local filesystem.​** **Syntax: hdfs dfs -get /hdfspath /localpath**   
**●​ du: Shows size of files/directories.​**  
 **Syntax: hdfs dfs -du /dirname**   
**●​ rm: Removes a file from HDFS.​**  
 **Syntax: hdfs dfs -rm /path**   
**●​ rmdir: Removes a directory in HDFS.​**  
 **Syntax: hdfs dfs -rmdir /path**   
**●​ mv: Moves or renames files/directories within HDFS.​**  
 **Syntax: hdfs dfs -mv /source /destination**   
**●​ cp: Copies files/directories within HDFS.​**  
 **Syntax: hdfs dfs -cp /source /destination**   
**●​ appendToFile: Appends data to an existing HDFS file.​**  
 **Syntax: hdfs dfs -appendToFile - /path**   
**●​ count: Counts files, directories, and bytes under a path.​**  
 **Syntax: hdfs dfs -count /path**   
**●​ tail: Shows the last part of a file.​**  
 **Syntax: hdfs dfs -tail /path**   
**●​ head: Shows the first part of a file.​**  
 **Syntax: hdfs dfs -head /path**   
**●​ test: Checks file/directory status.**

**●​ -e tests if path exists**   
 **●​ -d tests if path is a directory**   
 **●​ -f tests if path is a file**   
 **●​ -z tests if file is empty (zero bytes)​**  
 **Syntax: hdfs dfs -test -e /path**   
**●​ moveFromLocal: Moves files from local filesystem to HDFS (deletes**  **local after move).​**  
 **Syntax: hdfs dfs -moveFromLocal /local\_source**   
 **/hdfs\_destination**   
**●​ getmerge: Merges multiple files in HDFS into a single local file.​** **Syntax: hdfs dfs -getmerge -nl /source /localdestination ●​ checksum: Retrieves checksum of a file to verify data integrity.​** **Syntax: hdfs dfs -checksum /file**   
**●​ chgrp: Changes group ownership of files/directories.​**  
 **Syntax: hdfs dfs -chgrp new\_group /file**

**●​ stat: Shows last modification time of a file/directory.​**  
 **Syntax: hdfs dfs -stat /path**   
**●​ expunge: Empties the HDFS trash, permanently deleting files.​** **Syntax: hdfs dfs -expunge**   
**●​ chown: Changes owner and group of files/directories.​**  
 **Syntax: hdfs dfs -chown owner:group /path**   
**●​ chmod: Changes permissions of files/directories.​**  
 **Syntax: hdfs dfs -chmod 777 /path**​  
 **(Permissions: Read=4, Write=2, Execute=1)**   
**●​ setrep: Sets replication factor for files/directories.​**  
 **Syntax: hdfs dfs -setrep -w <n> /path**​  
 **(-w waits for replication to complete, n is replication count)**

**​**

***HIVE COMMANDS***

**●​ Hive is a data warehousing infrastructure built on Hadoop, enabling SQL-like**  **querying (HiveQL) over large datasets stored in HDFS.**

**●​ It converts HiveQL queries into MapReduce jobs for batch processing of big**  **data.**

**●​ Hive is designed for batch processing, not real-time or OLTP workloads.**

**HiveQL Command Categories:**

**●​ DDL (Data Definition Language): CREATE, DROP, ALTER, TRUNCATE ●​ DML (Data Manipulation Language): INSERT, SELECT, UPDATE, DELETE ●​ Partitioning: Efficiently manages large datasets by dividing tables into**  **partitions.**

**●​ Joins: Supports INNER, LEFT, RIGHT, FULL OUTER joins.**

**●​ Aggregation: COUNT, SUM, AVG, MIN, MAX functions.**

**●​ Sorting & Filtering: ORDER BY, WHERE, HAVING clauses.**

**Database Operations:**

**●​ Create database: CREATE DATABASE ecommerce\_db; ●​ Use database: USE ecommerce\_db;**   
**●​ Show databases: SHOW DATABASES;**   
**●​ Drop database: DROP DATABASE ecommerce\_db CASCADE;**

**Table Operations:**

**●​ Create table example:**   
**●​ sql**

**CREATETABLE products (**   
 **product\_id INT,**   
 **name STRING,**   
 **category STRING,**

|  |
| --- |
| **price FLOAT** |

**)**   
**ROW FORMAT DELIMITED**   
**FIELDSTERMINATEDBY','**   
**STORED AS TEXTFILE;**

|  |
| --- |
| **●​** |

**●​ Show tables: SHOW TABLES;**   
**●​ Describe table: DESCRIBE products;**   
**●​ Drop table: DROP TABLE products;**

**Data Operations:**

**●​ Insert data into table:**   
 **●​ sql**   
**INSERTINTOTABLE products VALUES (1, 'Laptop', 'Electronics', 800.50), (2, 'Smartphone', 'Electronics', 500.75);**

|  |
| --- |
| **●​** |

**●​ Load data from HDFS:**   
 **●​ sql**   
**LOADDATA INPATH '/user/hive/products.csv'INTOTABLE products;**

|  |
| --- |
| **●​** |

**●​ Select data: SELECT \* FROM products;**   
 **●​ Filter data: SELECT \* FROM products WHERE category = 'Electronics'; Joins in HiveQL:**   
 **●​ Inner Join:**   
 **●​ sql**   
**SELECT p.name, s.amount FROM products p JOIN sales s ON**   
**p.product\_id = s.product\_id;**

|  |
| --- |
| **●​** |

**●​ Left Join, Right Join, Full Outer Join supported similarly.**

**Aggregation and Sorting:**   
 **●​ Group By example:**   
 **●​ sql**   
**SELECT category, COUNT(\*) AS product\_count FROM products GROUP BY category;**

|  |
| --- |
| **●​** |

**●​ Order By example:**   
 **●​ sql**   
**SELECT \* FROM products ORDERBY price DESC;**

|  |
| --- |
| **●​** |

**●​ Limit example: SELECT \* FROM products LIMIT 5; Views:**   
 **●​ Create view:**   
 **●​ sql**

**CREATEVIEW electronics\_view ASSELECT \* FROM products WHERE category = 'Electronics';**

|  |
| --- |
| **●​** |

**●​ Use and drop views supported.**

**Operators in HiveQL:**

**●​ Arithmetic: +, -, \*, /, %**   
**●​ Comparison: =, !=, >, <, >=, <=**   
**●​ Logical: AND, OR, NOT**   
**●​ String: LIKE, REGEXP, IN**

**Complex Data Types:**

**●​ Arrays, Maps, Structs supported for complex data modeling. ●​ Example:**   
**●​ sql**

**CREATETABLE temperature (**   
  **sno INT,**   
  **place STRING,**   
  **mytemp ARRAY<DOUBLE>**   
**)**   
**ROW FORMAT DELIMITED**   
**FIELDSTERMINATEDBY'\t'**

**●​ COLLECTION ITEMS TERMINATEDBY',';**

**Hive Data Types:**

**●​ *Primitive Types:* TINYINT (1 byte), SMALLINT (2 bytes), INT (4 bytes), FLOAT**  **(4 bytes), DOUBLE (8 bytes)**   
**●​ *Date/Time Types:* TIMESTAMP (with nanosecond precision), DATE**   
 **(YYYY-MM-DD format)**   
**●​ *String Types:* STRING (sequence of characters), VARCHAR (1 to 65535**  **length), CHAR (fixed length up to 255)**   
**●​ *Complex Types:* Arrays (homogeneous collections), Maps (key-value pairs**  **with primitive keys), Structs (nested complex structures)**

**Hive Bucketing:**

**●​ Improves query performance by hashing data into buckets using CLUSTERED**  **BY clause.**

**●​ Buckets determine the number of files and ensure even data distribution. ●​ Often used together with partitioning for faster queries.**

**●​ Example to create bucketed table:**   
**●​ sql**

**CREATETABLE table\_name (column1 DATA\_TYPE, column2 DATA\_TYPE) CLUSTEREDBY (column\_name) INTO N BUCKETS**   
**STORED AS FILE\_FORMAT;**

**●​ Enable bucketing with SET hive.enforce.bucketing = true;**

**Hive Partitioning:**

**●​ Splits data into partitions based on a partition column to reduce scan times**  **and improve query performance.**

**●​ Supports static and dynamic partitioning.**

**●​ Example to create partitioned table:**   
**●​ sql**

**CREATETABLE table\_name (column1 DATA\_TYPE, column2 DATA\_TYPE) PARTITIONED BY (partition\_column DATA\_TYPE)**   
**STORED AS FILE\_FORMAT;**

**●​ Insert data into partition:**   
**●​ sql**

**INSERTINTO table\_name PARTITION (partition\_column=value) VALUES (value1, value2);**

|  |
| --- |
| **●​** |

**Basic Hive Operations:**

**●​ Create database: CREATE DATABASE IF NOT EXISTS abc; ●​ Show databases: SHOW DATABASES;**   
**●​ Create table with delimiter and storage format:**   
**●​ sql**

**CREATETABLE customers(id INT, fname STRING, lname STRING, city STRING)**   
**ROW FORMAT DELIMITED**   
**FIELDSTERMINATEDBY'|'**   
**STORED AS TEXTFILE;**

**●​ Load data into table:**   
**●​ sql**

**LOADDATALOCAL INPATH 'path\_to\_file'INTOTABLE customers;**

**●​ Query data: SELECT \* FROM table;**   
**●​ Drop table: DROP TABLE table\_name;**   
**●​ Rename table: ALTER TABLE old\_name RENAME TO new\_name;**   
**●​ Add columns: ALTER TABLE table\_name ADD COLUMNS (salary INT);**

***HBASE COMMANDS***

**Apache HBase is an open-source, distributed, scalable, column-oriented NoSQL database built on top of the Hadoop Distributed File System (HDFS). It is modeled after Google's Bigtable and designed to provide random, real-time read/write access to very large datasets, often spanning billions of rows and millions of columns.**

**Key Features:**

**●​ Linear and modular scalability to handle massive data growth.**

**●​ Strictly consistent reads and writes for reliable data operations.**

**●​ Automatic sharding (region splitting) and load balancing across cluster**  **nodes.**

**●​ High availability with automatic failover support for RegionServers. ●​ Integration with Hadoop ecosystem and MapReduce for batch processing. ●​ Provides easy-to-use Java APIs and supports REST and Thrift gateways.**

**●​ Supports block caching and Bloom filters for fast query performance.**

**Architecture Components:**

**●​ HMaster: Manages cluster coordination, schema changes, region**  **assignments, and load balancing.**

**●​ RegionServers: Store and manage data regions, handle client read/write**  **requests, and perform background compactions.**

**●​ ZooKeeper: Coordinates distributed cluster state, leader election, and**  **failover processes.**

**●​ HDFS: Underlying storage layer providing fault tolerance and replication.**

**Use Cases:**

**●​ Suitable for applications requiring fast random access to large-scale**  **structured or semi-structured data.**

**●​ Used by companies like Facebook, Twitter, Yahoo, and Adobe for heavy write**  **and real-time data access workloads.**

**Limitations:**

**●​ Requires careful upfront schema design; lacks ad-hoc query flexibility. ●​ Does not support joins or full ACID transactions across multiple regions. ●​ Not optimized for small datasets or complex relational queries.**

**Comparison with HDFS:**

**●​ HDFS is a distributed file system optimized for high-throughput batch**  **processing and sequential data access.**

**●​ HBase provides low-latency random read/write access on top of HDFS,**  **enabling real-time querying of big data.**

**Creating tables and adding data to it:**   
**a. create 'person','data' //create table**   
**b. put 'person',1,'data:name','Sahil' //put data into table c. put 'person',1,'data:city','washington'**   
**d. put 'person',1,'data:id','10'**   
**e. scan 'person' // to print the data of person table f. get 'person','1' //get data from a particular row**   
**g. get 'person','1',{COLUMN => 'data:id'} //fetch data of particular column in a column family**   
**h. get 'person','1',{COLUMN => ['data:id','data:name']} //fetch data from multiple columns in a column family**

**Alter table:**   
**a. alter ‘table1’, {NAME => ‘COLFAM2’}//Adding column family**   
**b. alter ‘table1’,{NAME => ‘COLFAM2’ , METHOD =>**   
**‘delete’} // DELETING A COLUMN FAMILY**   
**c. alter ‘table1’,{NAME => ‘COLFAM2’ , VERSIONS => 2}//CHANGING THE VERSIONS OF THE COLUMN**   
**FAMILY**   
**d. alter 'student', READONLY //for making table read-only e. alter 'student', {NAME => 'semsester', VERSIONS => 5} f. alter 'student', MAXFILESIZE='65165'**   
**g. alter\_status 'student' ///how many regions of the table have been altered**   
**h. create\_namespace 'mynamespace' //creates a name space(logical grouping) of the tables in hbase**   
**i. create 'mynamespace:table\_one', 'colfam1' //create a table inside a namespace**

**j. describe\_namespace 'mynamespace' //describe**   
**namespace**   
**k. list\_namespace //list all namespaces**   
**​**  
**​**

***PIG COMMANDS***

**Apache Pig is a high-level platform and scripting language designed to analyze and process large datasets on Hadoop. It provides a language called Pig Latin, which simplifies writing data analysis programs by abstracting complex MapReduce jobs into easier-to-write scripts.**

**Key Features:**

**●​ Ease of Programming: Pig Latin is a procedural, SQL-like language that**  **enables programmers to express data transformations as data flows, making**  **it easier to write, understand, and maintain complex data processing tasks. ●​ Abstraction over MapReduce: Pig scripts are automatically compiled into a**  **series of MapReduce jobs, allowing users to focus on data logic rather than**  **low-level programming.**

**●​ Optimization: The Pig framework optimizes the execution plan automatically**  **to improve performance.**

**●​ Extensibility: Users can write their own functions (UDFs) for custom**  **processing.**

**●​ Supports Semi-structured Data: Pig can handle both structured and semi-structured data, supporting complex data types like tuples, bags, and maps.**

**Architecture Components:**

**●​ Parser: Parses Pig Latin scripts and generates a logical plan represented as**  **a directed acyclic graph (DAG).**

**●​ Optimizer: Optimizes the logical plan to reduce resource usage and**  **execution time.**

**●​ Compiler: Converts the optimized plan into MapReduce jobs.**

**●​ Execution Engine: Executes the MapReduce jobs on a Hadoop cluster and**  **returns results.**

**Use Cases:**

**●​ Ideal for data transformation, ETL (Extract, Transform, Load) tasks, and**  **prototyping MapReduce jobs without writing Java code.**

**●​ Useful when working with large volumes of data that require parallel**  **processing.**

**Comparison with Hive:**

**●​ Pig is procedural and better suited for data pipelines and transformations.**

**●​ Hive uses a declarative SQL-like language (HiveQL) and is preferred for**  **ad-hoc querying and reporting on structured data.**

**●​ Pig generally offers more control over data flow and can be faster for certain**  **operations.**

**Starting Apache Pig**

**●​ Command (Windows & Mac):**   
**●​ bash**

**pig -x local**

|  |
| --- |
| **●​** |

**●​ Opens the Grunt shell for interactive Pig Latin commands.**

**Basic Pig Latin Commands**

**●​ Load Data from Local File:**   
**●​ text**

**data = LOAD 'C:/Users/gurvi/Downloads/student.txt' USING PigStorage(',') AS (id:int, name:chararray, marks:int);**

|  |
| --- |
| **●​** |

**●​ *(On Mac, use Unix-style path, e.g., /Users/gurvi/Downloads/student.txt)* ●​ Display Data:**   
**●​ text**

**DUMP data;**

|  |
| --- |
| **●​** |

**●​ Store Data to Local Directory:**   
**●​ text**

**STORE data INTO 'C:/Users/gurvi/Documents/my\_output' USING PigStorage(',');**

|  |
| --- |
| **●​** |

**●​ *(On Mac, use Unix-style path)***

**Data Filtering**

**●​ Filter rows based on condition:**

**●​ text**   
**high\_scorers = FILTER data BY marks > 70; DUMP high\_scorers;**

|  |
| --- |
| **●​** |

**●​ Filter with multiple conditions:**   
 **●​ text**   
**mid\_range = FILTER data BY marks >= 60 AND marks <= 80; DUMP mid\_range;**

|  |
| --- |
| **●​** |

**●​ Filter by string equality:**   
 **●​ text**   
**john\_data = FILTER data BY name == 'John'; DUMP john\_data;**

|  |
| --- |
| **●​** |

**Sorting Data**   
 **●​ Sort ascending:**   
 **●​ text**   
**sorted\_data = ORDER data BY marks ASC;**   
**DUMP sorted\_data;**

|  |
| --- |
| **●​** |

**●​ Sort descending:**   
 **●​ text**   
**sorted\_data\_desc = ORDER data BY marks DESC; DUMP sorted\_data\_desc;**

|  |
| --- |
| **●​** |

**Handling NULL Values**   
 **●​ Remove rows where marks is NULL:**   
 **●​ text**   
**valiILTER data BY marks IS NOT NULL;**   
**d\_marks = F**  
 **●​ Remove rows where name or marks is NULL:**

**●​ text**

**clean\_data = FILTER data BY name IS NOT NULL AND marks IS NOT NULL;**

|  |
| --- |
| **●​** |

**●​ Replace NULL values:**   
**●​ text**

**cleaned = FOREACH data GENERATE**   
 **id,**

|  |
| --- |
| **(name IS NOT NULL ? name : 'Unknown') AS name,** |

|  |
| --- |
| **(marks IS NOT NULL ? marks : 0) AS marks;** |

**DUMP cleaned;**

|  |
| --- |
| **●​** |

**Grouping and Aggregation**

**●​ Group data by marks:**   
**●​ text**

**grouped\_data = GROUP data BY marks;**

|  |
| --- |
| **●​** |

**●​ Count students per marks group:**   
**●​ text**

**counted = FOREACH grouped\_data GENERATE group AS marks, COUNT(data) AS student\_count;**   
**DUMP counted;**

|  |
| --- |
| **●​** |

**Joining Data**

**●​ Load two datasets:**   
**●​ text**

**students = LOAD 'C:/Users/gurvi/Downloads/students.txt' USING PigStorage(',') AS (id:int, name:chararray);**   
**scores = LOAD 'C:/Users/gurvi/Downloads/scores.txt' USING PigStorage(',') AS (id:int, marks:int);**

|  |  |
| --- | --- |
| |  | | --- | | **●​** | |

**●​ Join on student ID:**   
**●​ text**

**joined\_data = JOIN students BY id, scores BY id;**   
**final = FOREACH joined\_data GENERATE students::id, name, marks;**   
**DUMP final;**

|  |
| --- |
| **●​** |

**Field Access and Type Casting**

**●​ Access first field:**   
**●​ text**

**ids = FOREACH data GENERATE $0;**

|  |
| --- |
| **●​** |

**●​ Cast fields to specific types:**   
**●​ text**

**converted = FOREACH data GENERATE**   
 **(int)$0 AS id,**

|  |
| --- |
| **(chararray)$1 AS name,** |

**(int)$2 AS marks;**