# BIG DATA
# PRACTICAL RECORD
# 2025

# List of Practicals

1. Install and configure Hadoop on a standalone system.
2. Execute basic Hadoop CLI commands for file operations.
3. Set up a Hadoop Distributed File System (HDFS) cluster.
4. Perform file operations in HDFS using CLI.
5. Understand and configure Namenode and Secondary Namenode.
6. Monitor Hadoop daemons and their functionalities.
7. Implement file block management and replication in HDFS.
8. Integrate Hadoop with other Big Data tools.
9. Troubleshoot common Hadoop errors and optimize performance.
10. Install and configure HBase in a Hadoop environment.
11. Start and stop HBase services using CLI.
12. Create an HBase table with column families.
13. Insert data into an HBase table using HBase shell.
14. Retrieve data from an HBase table using get and scan commands.
15. Update and delete data from an HBase table.
16. Perform bulk data import into HBase from HDFS.
17. Integrate HBase with MapReduce for large-scale data processing.
18. Implement CRUD operations in HBase using Java/Python API.
19. Configure HBase regions and understand region splitting.
20. Optimize HBase schema design for efficient querying.
21. Implement row key design strategies for faster access.
22. Configure HBase replication and backup.
23. Monitor HBase performance using administrative tools.
24. Implement security and access controls in HBase.
25. Use Apache Phoenix for SQL-based querying on HBase.
26. Integrate HBase with Hive and execute SQL queries.
27. Process HBase data using Apache Pig scripts.
28. Use Apache Sqoop to transfer data between RDBMS and HBase.
29. Implement real-time data ingestion into HBase using Apache Flume.
30. Set up and manage Apache ZooKeeper for HBase coordination.
31. Automate workflows using Apache Oozie with HBase.
32. Compare HBase with other NoSQL databases.
33. Write a basic MapReduce program for word count.
34. Execute a MapReduce job on a sample dataset.
35. Implement a join operation using MapReduce.
36. Use combiners and partitioners to optimize MapReduce jobs.
37. Analyze hardware and network topology for efficient MapReduce execution.
38. Integrate MapReduce with HBase for data processing.
39. Configure MapReduce for handling large-scale datasets.
40. Implement synchronization mechanisms in MapReduce.

## Before you start — recommended environment on Windows 11

- Use **WSL2 (Windows Subsystem for Linux 2)** with **Ubuntu 22.04** (or 20.04). WSL2 runs a real Linux kernel and is the easiest way to run Hadoop on Windows.
- If you prefer not to use WSL2, you can use **Docker Desktop (WSL2 backend)** to run Hadoop images, or attempt a **native Windows** install (requires winutils.exe and is error-prone). I strongly recommend **WSL2** for student labs.

If you already have WSL2/Ubuntu, skip the "Enable WSL" step below.

---

**A. Quick: Enable WSL2 & install Ubuntu (Windows 11)**

Open PowerShell **as Administrator** and run:

powershell

# Simple WSL install (Windows 11)

wsl --install -d Ubuntu-22.04

- Reboot when prompted.
- Launch **Ubuntu** from Start. Create a Linux account (choose hduser or any name; I'll use <your-username> in commands).
- Update packages inside Ubuntu:

bash

sudo apt update && sudo apt upgrade -y

If wsl --install is not available, enable features manually:

powershell

dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

wsl --set-default-version 2

the **exact steps** to get Hadoop running on your **Windows 11 laptop** using **WSL2 + Ubuntu 22.04**.
We'll go step-by-step so you can execute each command and verify before moving on.

---

## Practical 1 — Install and configure Hadoop on a standalone (pseudo-distributed) system on Windows 11 (via WSL2)

**Objective:** Run a single-node (pseudo-distributed) Hadoop (HDFS + YARN) on Windows 11 using WSL2/Ubuntu.

**Step 1 — Install Java & utilities (in Ubuntu WSL)**

Open Ubuntu shell and run:

```
sudo apt update
sudo apt install -y openjdk-11-jdk ssh rsync curl vim
java -version    # should show OpenJDK 11.x
```

**Step 2 — Create or use a user**

You can use the WSL user created earlier. Optionally create hduser:

```
sudo adduser hduser
sudo usermod -aG sudo hduser
su - hduser
```

(If you stay with your default user, use <your-username> in commands below.)

**Step 3 — Setup passwordless SSH (required by Hadoop scripts)**

```
ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
```

# Start SSH server so that 'ssh localhost' works

```
sudo apt install -y openssh-server
sudo service ssh start
ssh localhost   # should connect without asking password
exit
```

**Step 4 — Download & install Hadoop**

In WSL:

```
cd /tmp
HADOOP_VER=3.3.5            # you may choose a recent stable 3.x
wget https://downloads.apache.org/hadoop/common/hadoop-$HADOOP_VER/hadoop-$HADOOP_VER.tar.gz
sudo tar -xzf hadoop-$HADOOP_VER.tar.gz -C /usr/local
sudo mv /usr/local/hadoop-$HADOOP_VER /usr/local/hadoop
sudo chown -R $(whoami):$(whoami) /usr/local/Hadoop
```

**Step 5 — Set environment variables (add to ~/.bashrc)**

Append these lines to ~/.bashrc (use your actual java path):

```
echo 'export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64' >> ~/.bashrc
echo 'export HADOOP_HOME=/usr/local/hadoop' >> ~/.bashrc
echo 'export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin' >> ~/.bashrc
echo 'export HADOOP_MAPRED_HOME=$HADOOP_HOME' >> ~/.bashrc
echo 'export HADOOP_COMMON_HOME=$HADOOP_HOME' >> ~/.bashrc
echo 'export HADOOP_HDFS_HOME=$HADOOP_HOME' >> ~/.bashrc
echo 'export YARN_HOME=$HADOOP_HOME' >> ~/.bashrc
source ~/.bashrc
```

**Step 6 — Point Hadoop to Java**

Edit $HADOOP_HOME/etc/hadoop/hadoop-env.sh (set JAVA_HOME):

*# open in editor:*

```
vim $HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

*# set:*

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

**Step 7 — Minimal Hadoop configuration files**

Edit files in $HADOOP_HOME/etc/hadoop. Replace <your-username> with your WSL username.

**core-site.xml:**

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/home/<your-username>/hadoop_tmp</value>
  </property>
</configuration>
```

**hdfs-site.xml:**

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
```

```
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/home/<your-username>/hadoop_data/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/home/<your-username>/hadoop_data/datanode</value>
  </property>
</configuration>
```

**mapred-site.xml (create from template):**

```
cd $HADOOP_HOME/etc/hadoop
cp mapred-site.xml.template mapred-site.xml
```

**Edit mapred-site.xml:**

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

**yarn-site.xml:**

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>localhost</value>
  </property>
</configuration>
```

**Step 8 — Create directories and set permissions**

```
mkdir -p ~/hadoop_tmp
mkdir -p ~/hadoop_data/namenode
mkdir -p ~/hadoop_data/datanode
```

**Step 9 — Format the NameNode**

```
hdfs namenode -format
# look for: "NameNode formatted"
```

**Step 10 — Start HDFS and YARN**

```
$HADOOP_HOME/sbin/start-dfs.sh
$HADOOP_HOME/sbin/start-yarn.sh

# verify processes
jps

# expected output (some subset): NameNode, DataNode, SecondaryNameNode, ResourceManager, NodeManager, Jps
```

**Step 11 — Verify web UIs (open these in Windows browser)**

- NameNode UI: http://localhost:9870
- ResourceManager UI: http://localhost:8088

If localhost doesn't work, find WSL IP via ip addr show eth0 and use that IP.

**Expected quick HDFS verify:**

```
hdfs dfs -mkdir -p /user/<your-username>
hdfs dfs -put /etc/hosts /user/<your-username>/hosts_copy
hdfs dfs -ls /user/<your-username>
hdfs dfs -cat /user/<your-username>/hosts_copy | head
```

**Troubleshooting tips**

- If ssh localhost fails: ensure sudo service ssh start ran and ~/.ssh/authorized_keys permissions are 600.
- If jps not found: install openjdk-11-jdk (provides jps) and ensure PATH includes JDK bin.
- If daemons fail: check logs in $HADOOP_HOME/logs/.

## Practical 2 — Execute basic Hadoop CLI commands for file operations (HDFS commands inside WSL)

**Objective:** Practice basic HDFS file ops using hdfs dfs or hadoop fs.

Run these examples inside your WSL Linux shell.

**Create directory:**

```
hdfs dfs -mkdir -p /user/<your-username>/input
```

**Upload local file → HDFS:**

```
echo "Hello from WSL+Windows 11" > ~/hello.txt
hdfs dfs -put ~/hello.txt /user/<your-username>/input/
# or
hadoop fs -copyFromLocal ~/hello.txt /user/<your-username>/input/
```

**List directory:**

```
hdfs dfs -ls /user/<your-username>/input
```

**View file contents:**

```
hdfs dfs -cat /user/<your-username>/input/hello.txt
hdfs dfs -tail /user/<your-username>/input/hello.txt
```

**Download file (HDFS → local):**

```
hdfs dfs -get /user/<your-username>/input/hello.txt ~/hello_from_hdfs.txt
```

**Remove file/dir:**

```
hdfs dfs -rm /user/<your-username>/input/hello.txt
hdfs dfs -rm -r /user/<your-username>/input
```

**Check disk usage & health:**

```
hdfs dfs -du -h /user/<your-username>
hdfs dfsadmin -report
hdfs fsck / -files -blocks -locations
```

**Tips about using Windows files:**
To access a Windows file in WSL use /mnt/c/Users/.... Example:

```
hdfs dfs -put /mnt/c/Users/Student/Desktop/data.csv /user/<your-username>/
```

## Practical 3 — Set up a basic HDFS cluster (multi-node) for a Windows 11 lab

**Objective:** Show how to create a small multi-node HDFS cluster using multiple Windows 11 lab machines (each running WSL2).

**Important:** Each machine must run WSL2 Ubuntu and the same Hadoop version. This section gives the lab setup approach — do **not** copy the NameNode's dfs metadata across nodes (format NameNode only once).

**Two practical ways to do multi-node in a Windows lab:**

- **A. Multiple physical Windows 11 PCs** (each WSL2/Ubuntu): preferred for real multi-node labs.
- **B. Multi-node on single PC** using Docker images / VMs: useful if you have only one PC.

Below steps assume **option A**.

**Steps (A — using multiple Windows machines)**

1. **Install WSL2+Ubuntu+Hadoop** on each Windows PC following Practical 1 (same HADOOP_VER on all nodes).
2. **Network & hostname mapping**: on each WSL /etc/hosts add lines like:

```
192.168.1.10  namenode
192.168.1.11  datanode1
192.168.1.12  datanode2
```
(Replace IPs with actual lab network IPs.)

3. **Configure core-site.xml on ALL nodes** to point to NameNode:

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://namenode:9000</value>
</property>
```

4. **Configure hdfs-site.xml on all nodes**:

- On **NameNode** set dfs.namenode.name.dir (local dir).
- On **DataNodes** set dfs.datanode.data.dir (local dir).
- Set replication factor > 1 (e.g., 2):

```
<property>
  <name>dfs.replication</name>
  <value>2</value>
</property>
```

5. **List workers (DataNodes):** On NameNode edit $HADOOP_HOME/etc/hadoop/workers (or slaves) and include:

```
datanode1
datanode2
```

6. **Passwordless SSH**: On NameNode WSL:

```
ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa
ssh-copy-id <username>@datanode1
ssh-copy-id <username>@datanode2
ssh datanode1 'hostname'   # test
```

7. **Create data directories** on each host and set ownership:

```
sudo mkdir -p /home/<your-username>/hadoop_data/datanode
sudo chown -R <your-username>:<your-username> /home/<your-username>/hadoop_data
```

8. **Format NameNode** (only once, on the NameNode):

```
hdfs namenode -format
```

9. **Start HDFS (from NameNode)**:

```
$HADOOP_HOME/sbin/start-dfs.sh
```

This should start DataNode processes on the worker machines via SSH.

10. **Verify cluster**:

```
hdfs dfsadmin -report
```

# NameNode UI http://namenode:9870 shows live DataNodes

11. **Stop cluster**:

```
$HADOOP_HOME/sbin/stop-dfs.sh
```

**Notes & troubleshooting**

- Ensure the Windows firewall allows traffic between the IPs for Hadoop ports (9000, 9870, DataNode ports). In a lab network, open these ports or temporarily disable firewall (not recommended on public networks).
- If a DataNode doesn't register, check logs on that node ($HADOOP_HOME/logs/) and ensure SSH connectivity and correct hdfs-site paths.
- If replication is under-replicated, check hdfs dfsadmin -report and DataNode avail.

## Practical 4 — Perform advanced file operations in HDFS using CLI (in WSL)

**Objective:** Demonstrate useful HDFS operations beyond put/get (block inspection, replication change, fsck, concat, DistCp).

Examples:

1. **Upload directory of many files**

```
hdfs dfs -mkdir -p /user/<your-username>/data
hdfs dfs -put ~/data_samples/* /user/<your-username>/data/
```

2. **Inspect block locations**

```
hdfs fsck /user/<your-username>/data -files -blocks -locations
```

3. **Change replication factor for a file**

```
hdfs dfs -setrep -w 2 /user/<your-username>/data/largefile.dat
```

# -w makes the command wait for replication to finish

4. **Concatenate multiple HDFS files into one**

```
hdfs dfs -cat /user/<your-username>/data/part* | hdfs dfs -put - /user/<your-username>/data/combined.txt
```

5. **Use fsck to detect corrupt blocks**

```
hdfs fsck / -list-corruptfileblocks
```

6. **Copy between two clusters with DistCp** (demo only if you have two clusters):

```
hadoop distcp hdfs://sourceCluster:9000/user/data
hdfs://targetCluster:9000/user/data_copy
```

7. **Get file block size / replication metadata**

```
hdfs dfs -stat %o /user/<your-username>/data/largefile.dat   # shows block size (octal? varies)
hdfs dfs -ls -R /user/<your-username>                       # shows replication for files
```

8. **HDFS Trash vs delete**

- If fs.trash.interval is set in core-site.xml, deleted files go to Trash. Use hdfs dfs -rm -skipTrash to bypass trash.

**Verification**

- Check NameNode UI for file list and block distribution.
- hdfs dfs -du -h /user/<your-username> shows storage used.

## Practical 5 — Understand & configure NameNode and SecondaryNameNode (checkpointing) on Windows 11 (WSL2)

**Objective:** Teach the role of NameNode and SecondaryNameNode (SNN), configure checkpointing, and show how to view checkpoints.

**Concept recap**

- NameNode stores namespace metadata: fsimage and edits.
- As edits accumulate, to avoid huge edits files, **SecondaryNameNode** periodically fetches edits from the NameNode, merges with fsimage and writes a new checkpointed fsimage.
- **Important:** SNN is **not** a hot standby; it will not automatically replace a failed NameNode. For HA (Active/Standby), you need ZooKeeper + two NameNodes + ZKFC.

**Configure checkpoint directory & period (on NameNode's hdfs-site.xml)**

**Add:**

```
<property>
  <name>dfs.namenode.checkpoint.dir</name>
  <value>file:/home/<your-username>/hadoop_data/namesecondary</value>
</property>
<property>
  <name>dfs.namenode.checkpoint.period</name>
  <value>3600</value>  <!-- seconds between checkpoints (example) -->
</property>
```

**Create directory:**

**mkdir -p ~/hadoop_data/namesecondary**

**Restart HDFS daemons so SecondaryNameNode picks up config:**

**$HADOOP_HOME/sbin/stop-dfs.sh**
**$HADOOP_HOME/sbin/start-dfs.sh**
jps  # confirm SecondaryNameNode process present

**Verify checkpoint**

- Check **NameNode web UI** (http://localhost:9870) — it shows "Last checkpoint time" and "Time since last checkpoint".
- Inspect SecondaryNameNode logs:

```
tail -n 200 $HADOOP_HOME/logs/hadoop-$(whoami)-secondarynamenode-*.log

# look for messages: "Checkpointed" or "Merged edits"
```

## Practical 6 — Monitor Hadoop daemons and their functionalities

**Objective:** Learn how to monitor Hadoop services (NameNode, DataNode, ResourceManager, NodeManager, SecondaryNameNode) and check cluster health.

**Prerequisite:** Hadoop running in WSL (NameNode/DataNode/ResourceManager/NodeManager started).

**1) Quick process check (local)**

# list Java Hadoop processes

**jps**

# expected output includes: NameNode, DataNode, SecondaryNameNode, ResourceManager, NodeManager, Jps

**2) HDFS health & capacity**

# summary

**hdfs dfsadmin -report**

# check under/over-replicated and corrupt blocks

**hdfs fsck / -blocks -locations | head -n 50**
**hdfs fsck / -list-corruptfileblocks**

**What to look for:** live DataNodes, capacity used/free, under-replicated blocks = 0, corrupt blocks = 0.

**3) YARN / applications**

# list nodes known to ResourceManager

**yarn node -list**

# list applications

**yarn application -list**

# view application logs (replace <appId>)

**yarn logs -applicationId <applicationId> | tail -n 200**

**4) Web UIs (open in Windows browser)**

- NameNode: http://localhost:9870 — filesystem browser, live nodes.
- ResourceManager: http://localhost:8088 — apps, cluster metrics.
- DataNode UI (per node): http://<datanode-host>:9864 (WSL localhost if single node).

**5) Tail logs (for real-time monitoring)**

*# tail NameNode log*

**tail -f $HADOOP_HOME/logs/hadoop-$(whoami)-namenode-*.log**

*# tail DataNode*

**tail -f $HADOOP_HOME/logs/hadoop-$(whoami)-datanode-*.log**

*# tail ResourceManager*

**tail -f $HADOOP_HOME/logs/yarn-$(whoami)-resourcemanager-*.log**

**6) OS-level monitoring inside WSL**

*# CPU/memory*

**top** # or htop if installed

# disk IO

**iostat -x 1 5** # install sysstat: sudo apt install sysstat

# network connections

**ss -tulpn**

**7) JMX & metrics (optional)**

NameNode JMX endpoint: http://localhost:9870/jmx — use for automated monitoring or Prometheus exporters.

**Troubleshooting tips**

- If jps doesn't show a daemon: check logs in $HADOOP_HOME/logs/.
- If DataNode not listed in NameNode UI: check DataNode log for connectivity or permission issues.
- If ResourceManager shows unhealthy NodeManagers: check yarn.nodemanager.health-checker settings and NodeManager logs.

## Practical 7 — Implement file block management and replication in HDFS

**Objective:** Understand HDFS block-splitting, view block locations, change replication settings, simulate DataNode failure and rebalance.

**1) View current default blocksize & replication**

*# show configured value*
**hadoop fs -stat %o /** # block size maybe not shown for root; check config file instead
*# display effective dfs.blocksize and dfs.replication from config*
**grep -E "dfs.blocksize|dfs.replication" $HADOOP_HOME/etc/hadoop/hdfs-site.xml -n**

Default block size is often 128MB or 256MB; replication default 1 for single-node labs.

**2) Upload a large test file and inspect blocks**

# create a 200MB test file in WSL
dd if=/dev/zero of=~/largefile.dat bs=1M count=200

# put into HDFS
**hdfs dfs -put ~/largefile.dat /user/$(whoami)/**

# inspect blocks & locations
**hdfs fsck /user/$(whoami)/largefile.dat -files -blocks -locations**
**Expected:** File split into multiple blocks and list shows block IDs and DataNode host(s).

**3) Change replication factor for a file or directory**

# set replication to 2 for a file (works only if you have 2+ DataNodes)
**hdfs dfs -setrep -w 2 /user/$(whoami)/largefile.dat**

# set replication recursively for a directory
**hdfs dfs -setrep -R 2 /user/$(whoami)/**
-w waits until replication achieved. Use hdfs dfsadmin -report to verify replication distribution.

**4) Simulate DataNode failure & observe under-replicated blocks**

# stop DataNode on one node (WSL single node: simulate by stopping datanode process)
**$HADOOP_HOME/sbin/hadoop-daemon.sh stop datanode**

# check NameNode UI or run:
**hdfs dfsadmin -report | sed -n '1,120p'**

# check under-replicated blocks
**hdfs fsck / -under-replicated -files -blocks**
**Recovery:** restart DataNode or add new DataNode; HDFS will replicate missing blocks automatically.

**5) Rebalance cluster (after adding nodes or changing capacity)**

*# run the HDFS balancer with default threshold*
**hdfs balancer**

*# or specify threshold (percentage)*
**hdfs balancer -threshold 10**
**What it does:** Redistributes blocks across DataNodes to even out HDFS utilization.

**6) Detect corrupt blocks & attempt fix**

**hdfs fsck / -list-corruptfileblocks**

# to try to fix corrupt files, copy to a safe location, re-put from local, or replace missing block source if available

**Tuning tips**

- Increase block size for large datasets (e.g., 256MB or 512MB) to reduce metadata and improve throughput:
    - set dfs.blocksize in hdfs-site.xml or set per-file during upload using -Ddfs.blocksize=268435456 with hadoop fs -Ddfs.blocksize=268435456 -put ...
- Use a replication factor of 3 in production for fault tolerance; in lab with 1–2 nodes set to 1 or 2.