**Hadoop HDFS Shell Commands**

1. **mkdir**: Create the directory in HDFS.

*Usage:* ***hdfs dfs -mkdir [-p] <paths>***

*Options:*

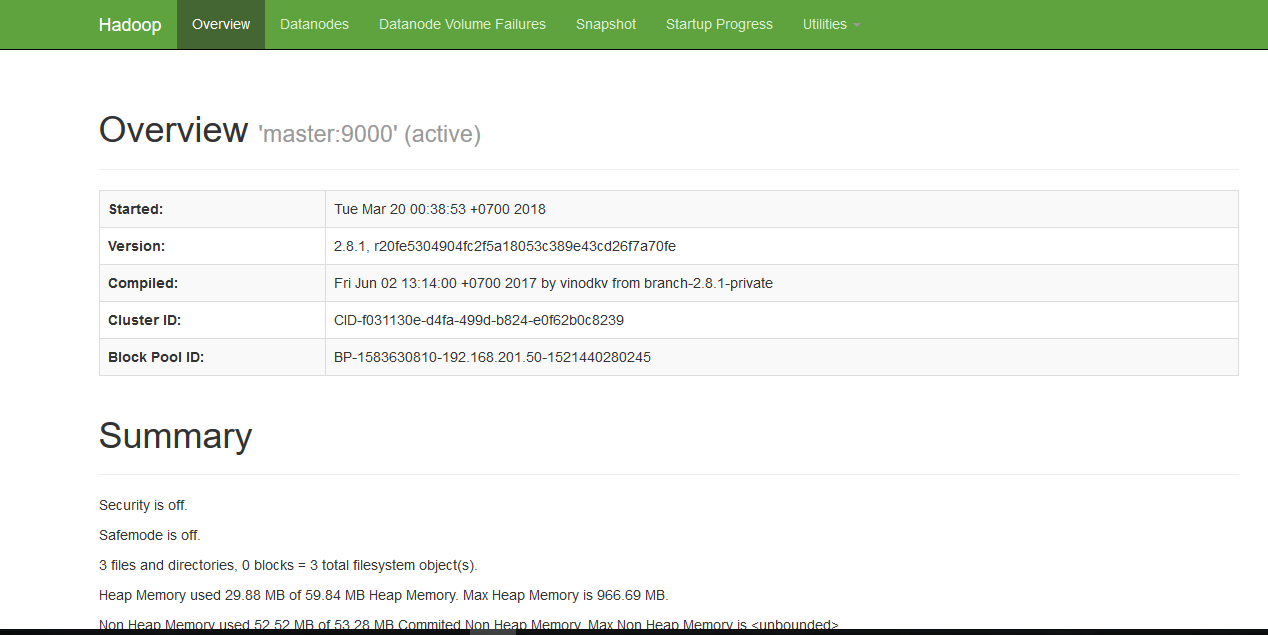
*The -p option behavior is much like Unix mkdir -p, creating parent directories along the path.*

Example:

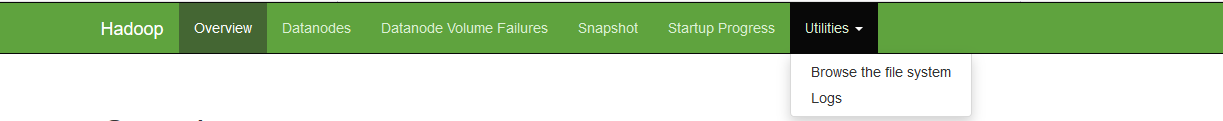
* hdfs dfs –mkdir /data
* hdfs dfs –mkdir hdfs://192.168.201.50:9000/result

Result:

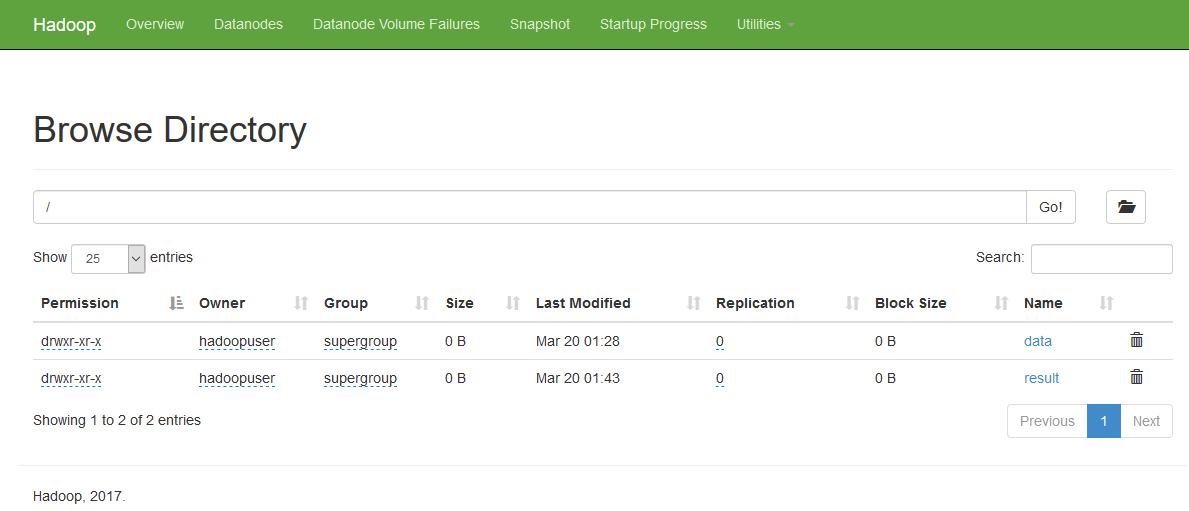
* Go to <http://192.168.201.50:50070> to check the result



* Click **Utilities** -> **Browse the file system**



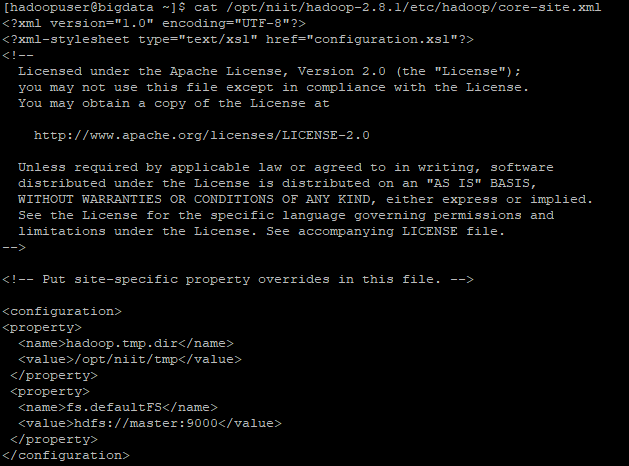
* As the result, we have two created folder.



☺ Notes:

The default address of namenode web UI is http://192.168.201.50:50070/. You can open this address in your browser and check the namenode information.

The default address of namenode server is hdfs://192.168.201.50:8020/. You can connect to it to access HDFS by HDFS api. But in the example above, we use port **9000** because we defined it in $HADOOP\_HOME/etc/hadoop/core-site.xml



Please refer <http://blog.cloudera.com/blog/2009/08/hadoop-default-ports-quick-reference/> for Hadoop default ports.

1. **ls:** List directories/files in HDFS

*Usage: hdfs dfs -ls [-R] <args>*

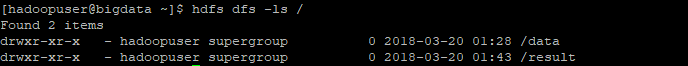
*Options:*

*-R option: list subdirectories recursively*

Example:

hdfs dfs –ls /

Result:



1. **put:** Copy file from single src, or multiple srcs from local file system to the destination file system.

*Usage: hdfs dfs -put <localsrc> ... <dst>*

Example:

hdfs dfs –put /home/hadoopuser/txnsl.txt /data

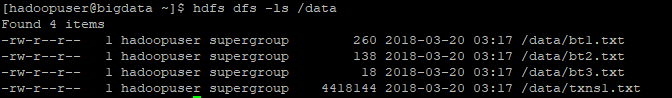
hdfs dfs –put /home/hadoopuser/bt1.txt /home/hadoopuser/bt2.txt /data

hdfs dfs –put /home/hadoopuser/bt3.txt hdfs://192.168.201.50:9000/data

Result:







1. **get**: Copy files from hdfs to the local file system.

*Usage: hdfs dfs -get <src> <localdst>*

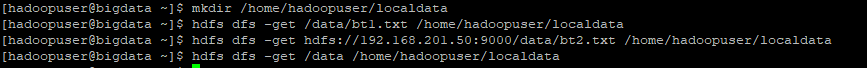
*Example:*

hdfs dfs –get /data/bt1.txt /home/hadoopuser/localdata

hdfs dfs –get hdfs://192.168.201.50:9000/data/bt2.txt /home/hadoopuser/localdata

hdfs dfs –get /data /home/haoopuser/localdata

*Result:*





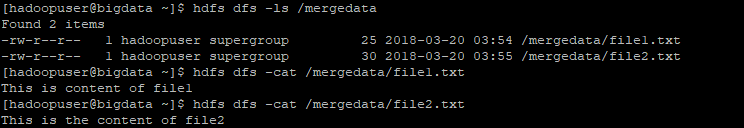
1. **getmerge**: Takes a source directory and a destination file as input and concatenates files in src into the destination local file.

*Usage: hdfs dfs -getmerge <src> <localdst>*

*Example:*

hdfs dfs –getmerge /mergedata /home/hadoopuser/mergedatalocal/merge

Result:





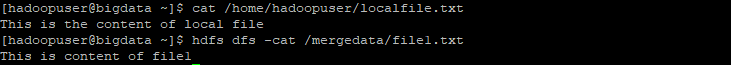
1. **appendToFile**: Append single src, or multiple srcs from local file system to the destination file system

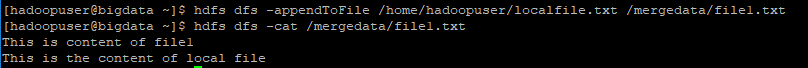
*Usage: hdfs dfs -appendToFile <localsrc> ... <dst>*

*Example:*

*hdfs dfs –appendToFile /home/hadoopuser/localfile.txt /mergedata/file1.txt*

*Result:*





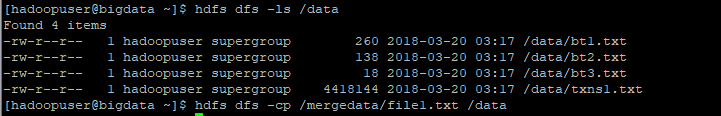
1. **cp**: Copy files from source to destination

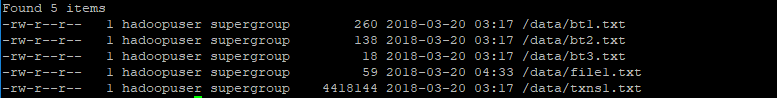
*Usage: hdfs dfs -cp <src> ... <dst>*

*Example:*

*hdfs dfs –cp /mergedata/file1.txt /data*

*Result:*





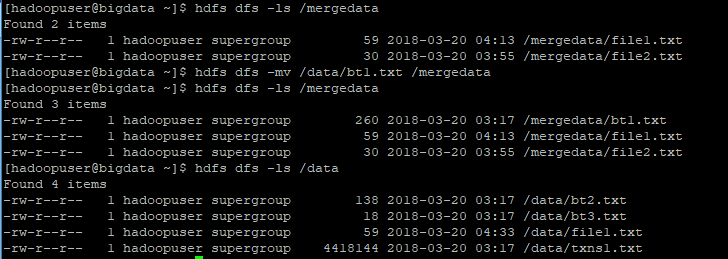
1. **mv**: Moves files from source to destination

*Usage: hdfs dfs -mv <src> ... <dst>*

*Example:*

*hdfs dfs –mv /data/bt1.txt /mergedata*

*Result:*



1. **du:** Displays sizes of files and directories

*Usage: hdfs dfs -du [-s] [-h] URI*

*Options:*

* + *The -s option will result in an aggregate summary of file lengths being displayed, rather than the individual files.*
  + *The -h option will format file sizes in a "human-readable" fashion (e.g 64.0m instead of 67108864)*

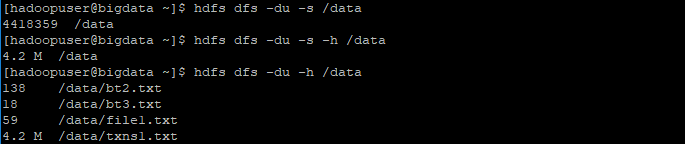
*Example:*

*hdfs dfs –du –s /data*

*hdfs dfs –du –s –h /data*

*hdfs dfs –du –h /data*

*Result:*



1. **rm**: Delete files specified as args.

*Usage: hdfs dfs -rm [-f] [-r |-R] [-skipTrash] [-safely] URI [URI ...]*

*Options:*

* *The -f option will not display a diagnostic message or modify the exit status to reflect an error if the file does not exist.*
* *The -R option deletes the directory and any content under it recursively.*
* *The -r option is equivalent to -R.*
* *The -skipTrash option will bypass trash, if enabled, and delete the specified file(s) immediately. This can be useful when it is necessary to delete files from an over-quota directory.*
* *The -safely option will require safety confirmation before deleting directory with total number of files greater than hadoop.shell.delete.limit.num.files (in core-site.xml, default: 100). It can be used with -skipTrash to prevent accidental deletion of large directories. Delay is expected when walking over large directory recursively to count the number of files to be deleted before the confirmation.*

*Example1:*

*hdfs dfs –rm /data/file1.txt*

* What is the hadoop trash feature?

The Hadoop trash feature helps prevent accidental deletion of files and directories. If trash is enabled and a file or directory is deleted, the file is moved to the **.Trash** directory in the user’s home directory instead of being deleted.

* How to enable Trash in hadoop?

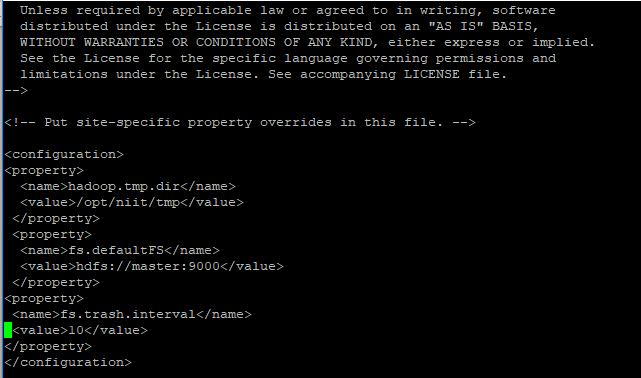
In [Hadoop](http://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/) to enable the trash feature and set the time delay for the trash removal, we can set the **fs.trash.interval** property in **core-site.xml** to the delay (in minutes).  
For example, if you want users to have 10 minutes) to restore a deleted file, you should specify following configuration parameter in the core-site.xml. Setting the value to 0 will disable the Trash feature.

***<property>***

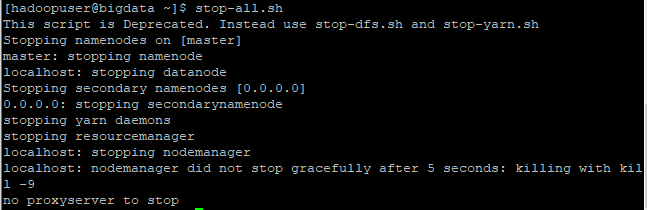
***<name>fs.trash.interval</name>***

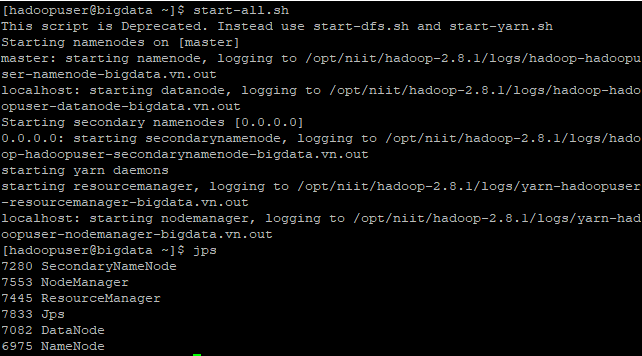
***<value>10</value>***

***</property>***

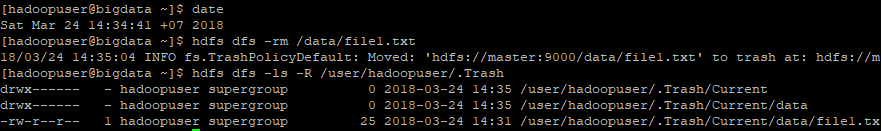


Then stop and start hadoop again.

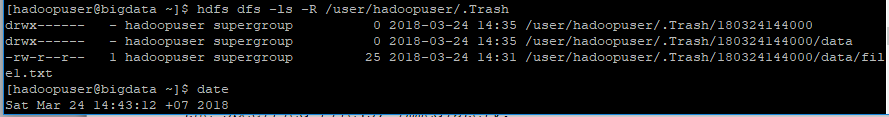




Try to delete /data/file1.txt. file1.txt is move to /user/hadoopuser/.Trash/Current/data. You can restore files and directories in the trash simply by moving them to a location outside the .Trash directory.



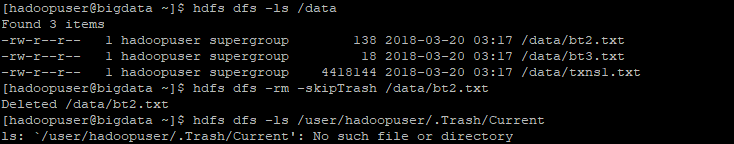
Wait for a little more than 10 mins and check again. file1.txt will be removed from HDFS permanently. But before removing permanently, it is moved to checkpoint folder at the point of time ~7 mins.





*Example1: The -skipTrash option will bypass trash, and delete the specified file(s) immediately.*

*hdfs dfs –rm –skipTrash /data/bt2.txt*



1. **expunge:** Permanently delete files in checkpoints older than the retention threshold from trash directory, and create new checkpoint.

*Usage: hdfs dfs –expunge*

When checkpoint is created, recently deleted files in trash are moved under the checkpoint. Files in checkpoints older than fs.trash.interval will be permanently deleted on the next invocation of -expunge command.

If the file system supports the feature, users can configure to create and delete checkpoints periodically by the parameter stored as fs.trash.checkpoint.interval (in core-site.xml). This value should be smaller or equal to fs.trash.interval.

Example:

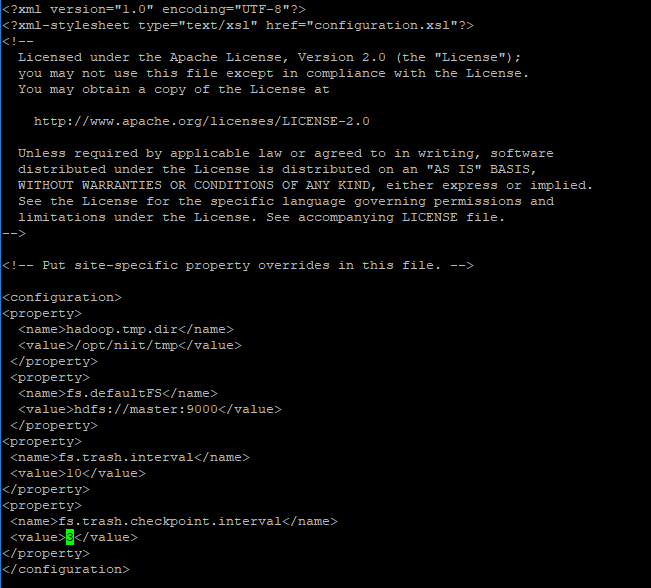
* Add the following configuration parameter in core-site.xml

***<property>***

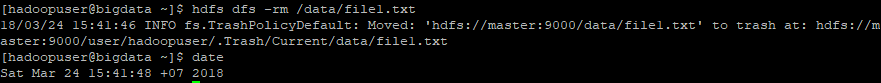
***<name>fs.trash.checkpoint.interval</name>***

***<value>3</value>***

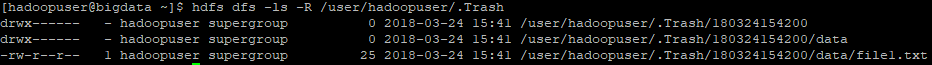
***</property>***



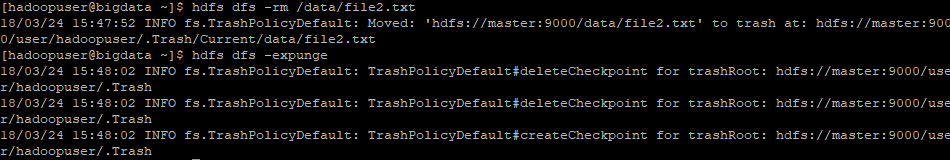
* Delete one file in HDFS



* After about 3 mins (fs.trash.checkpoint.interval), it is moved to checkpoint folder



* If we want it to move immediately into checkpoint folder, we enter the command: hdfs dfs –expunge



* Check the result in .Trash

