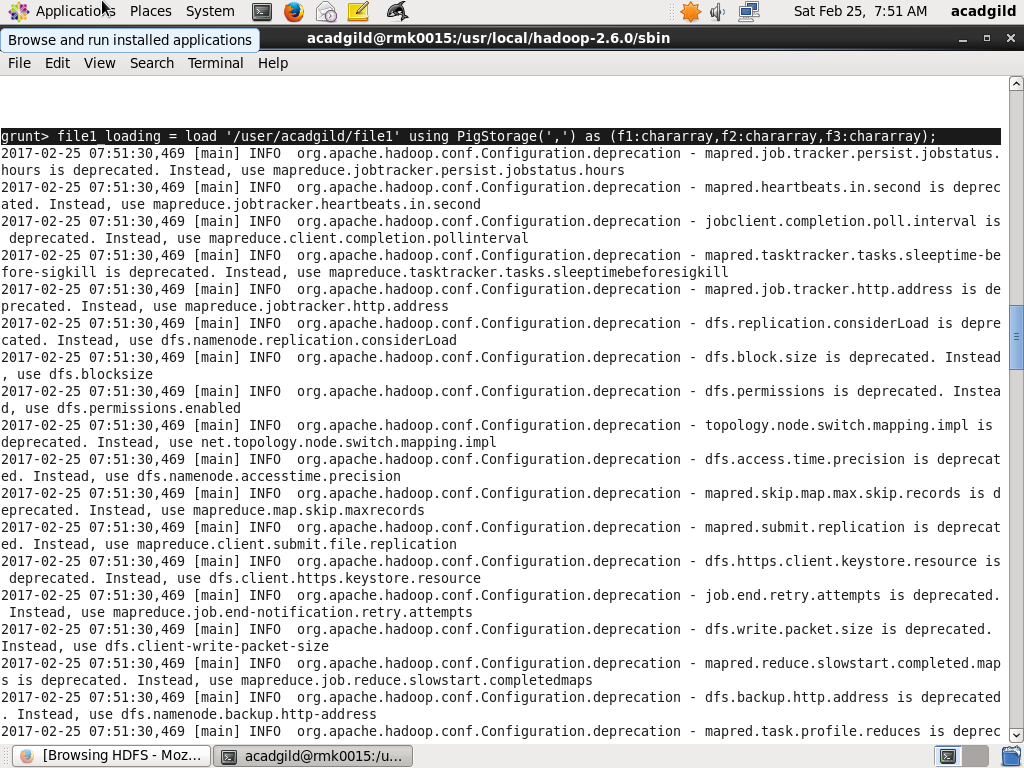
**Concat:**

Concatenates two fields of type chararray or two fields of type bytearray.

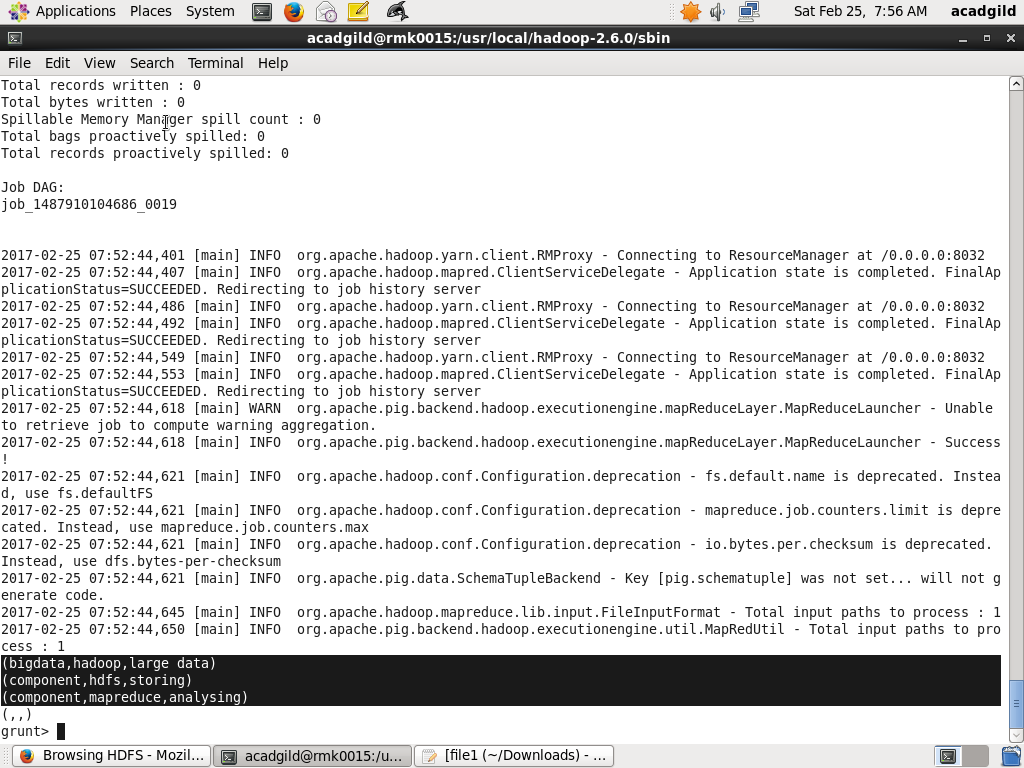
Syntax:

CONCAT (expression, expression)

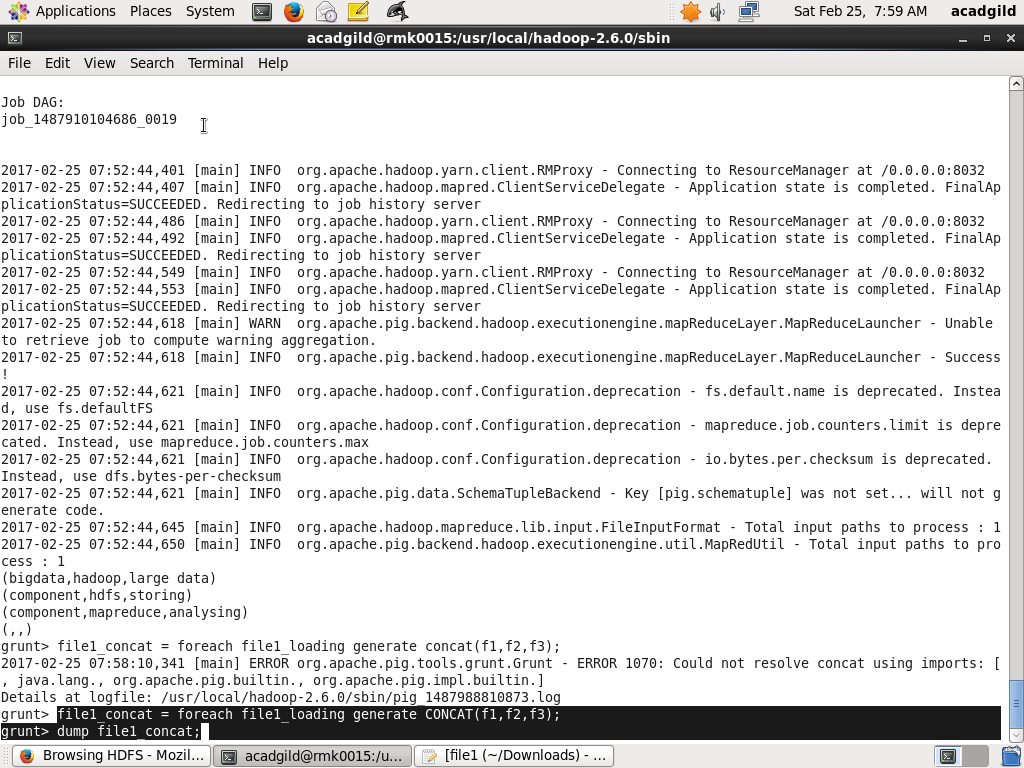
Loading the file1 file into file1\_loading:

****

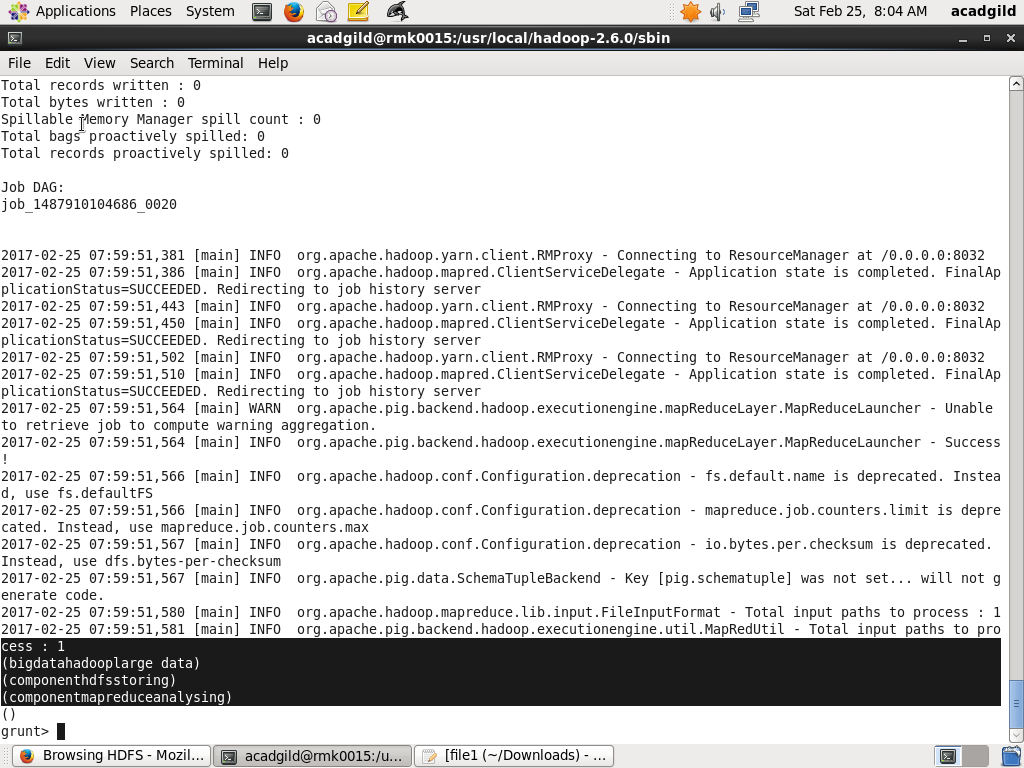
Dump the file1\_loading:



Apply the concat operation into file1\_loading and give the name as file1\_concat:



Dump the file1\_concat:



**TOKENIZE:**

Splits a string and outputs a bag of words.

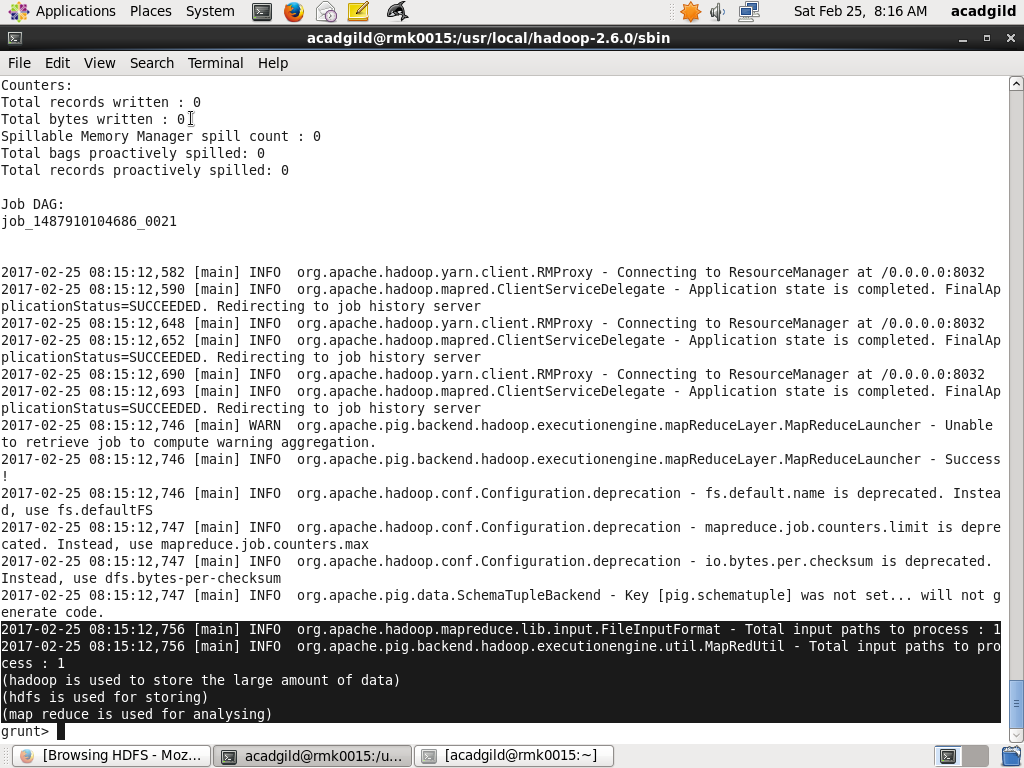
Syntax:

TOKENIZE(expression)

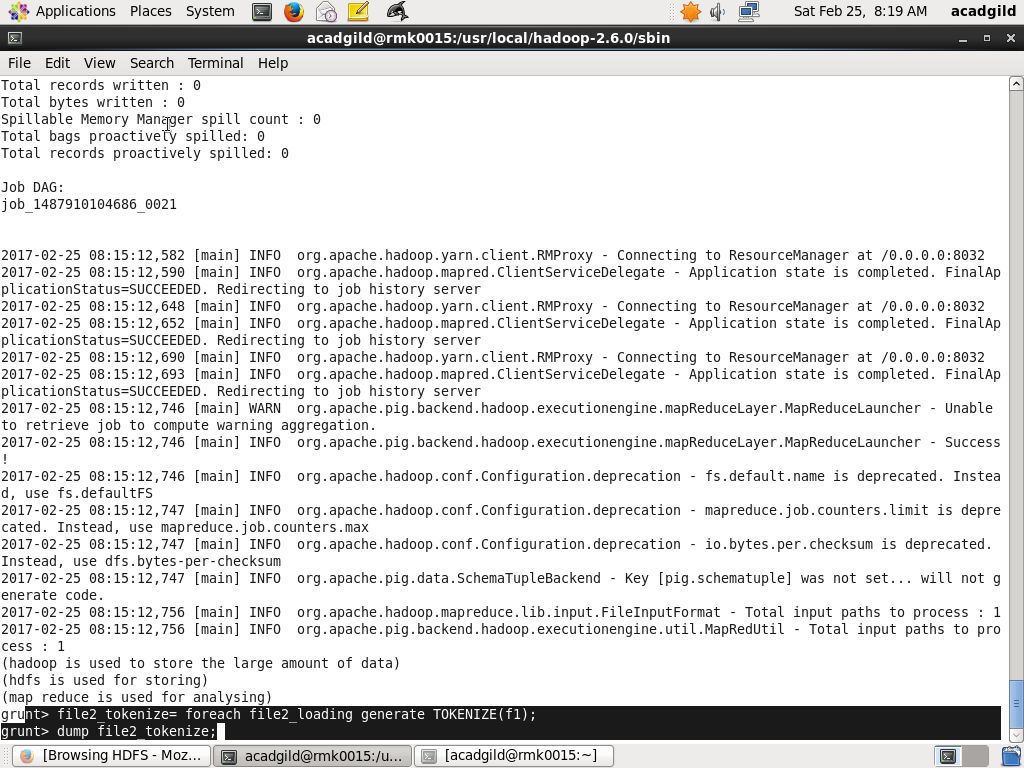
Loading the file2 into file2\_loading:

****

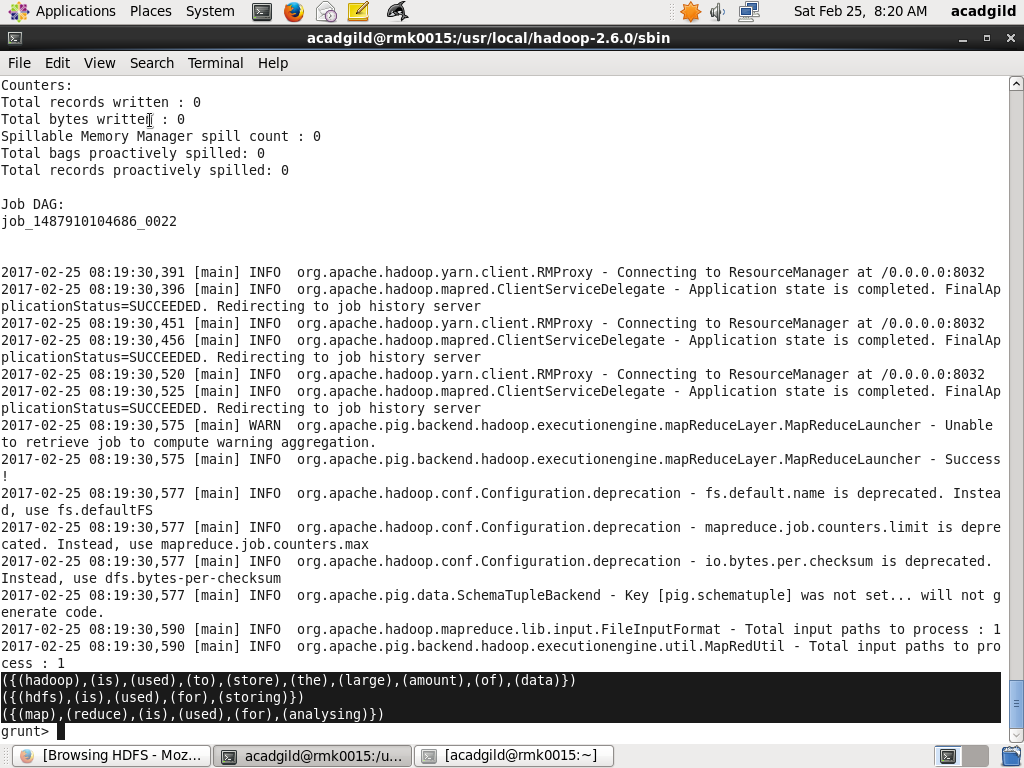
Dump the file2\_loading:



Apply the TOKENIZE command into file2\_loading and the name as file2\_tokenize:



Dump the file2\_tokenize:



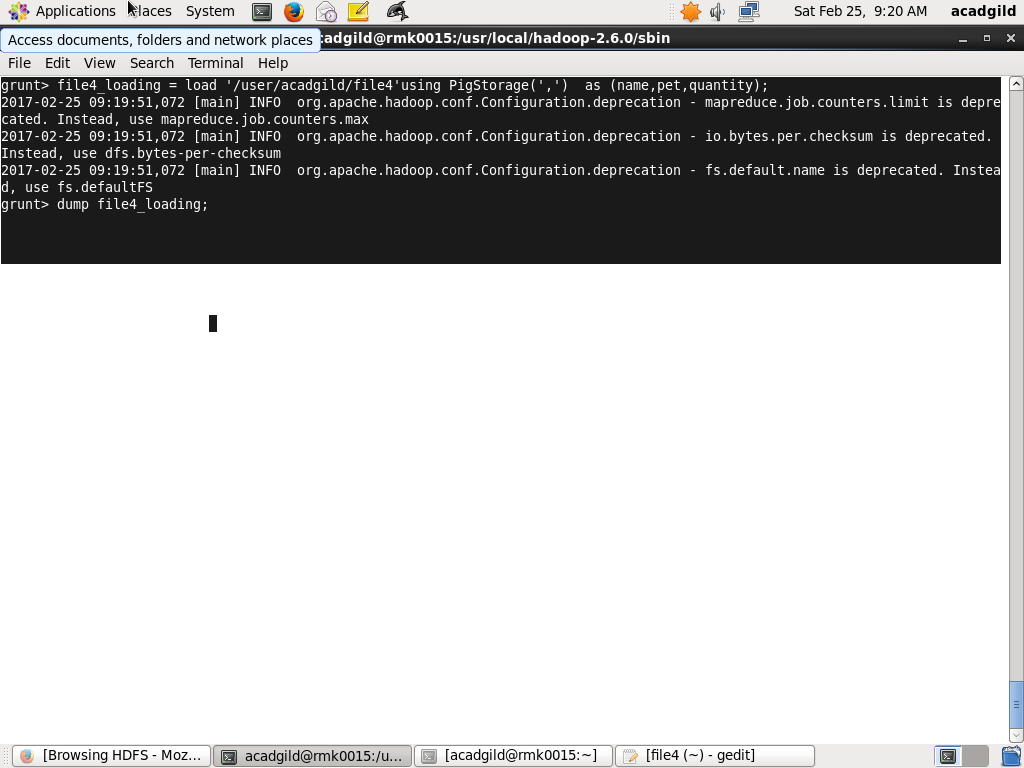
**SUM:**

Computes the sum of the numeric values in a single-column bag. SUM requires a preceding GROUP ALL statement for global sums and a GROUP BY statement for group sums.

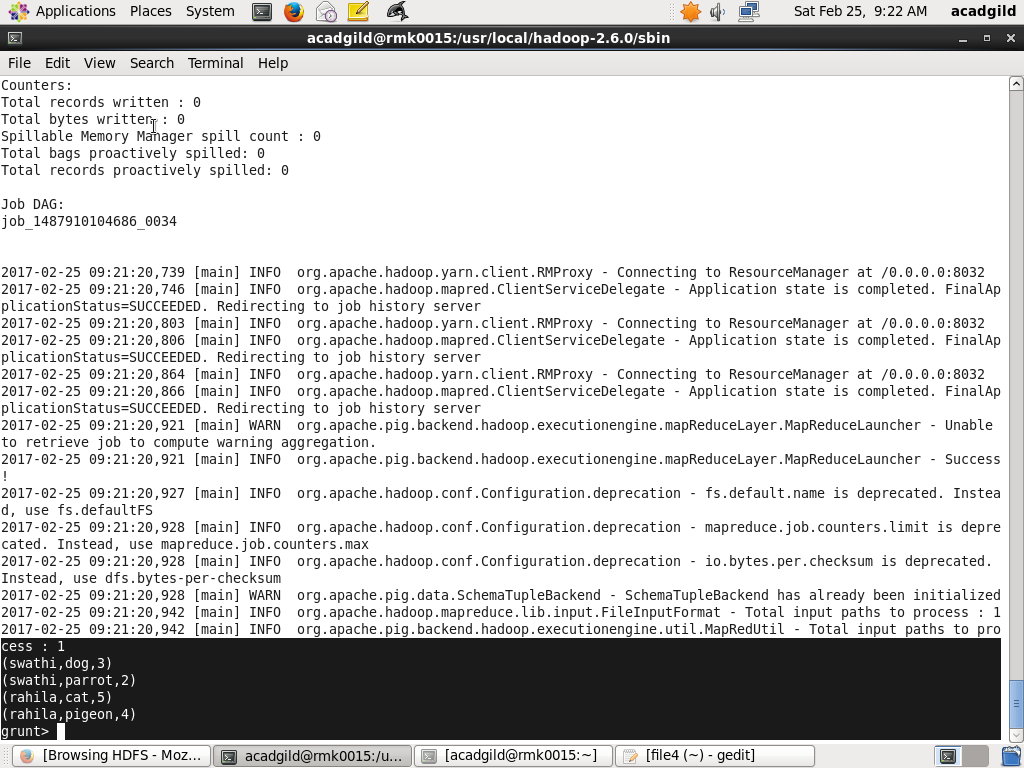
Syntax:

SUM(expression)

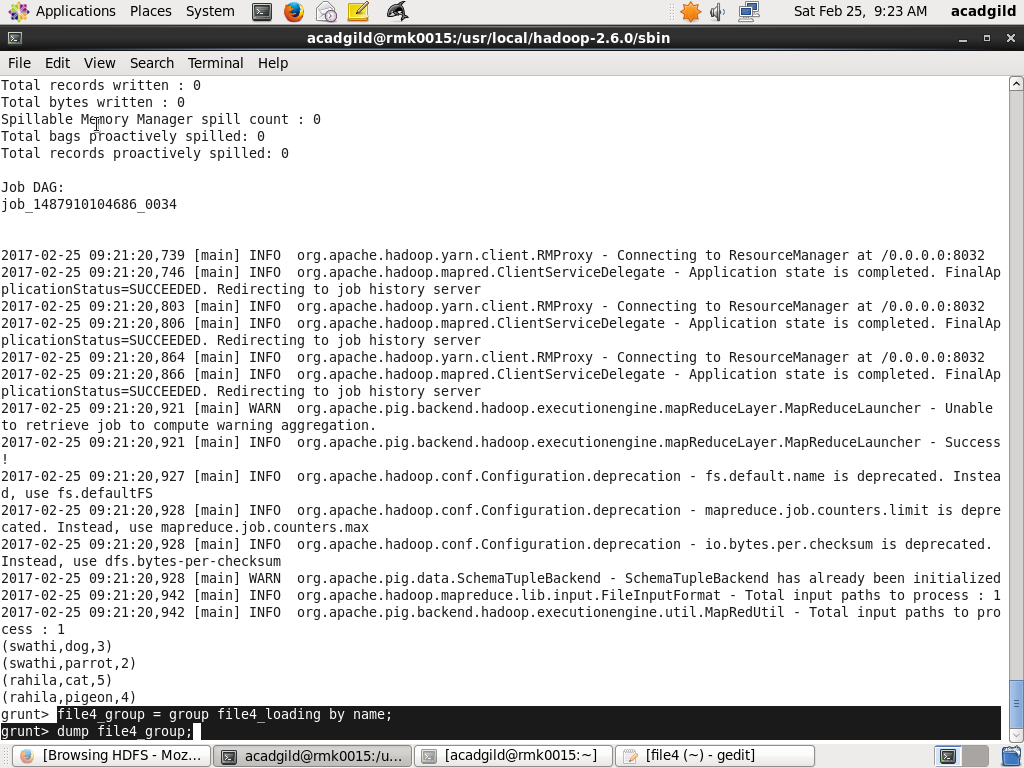
Load the file4 into file4\_loading:

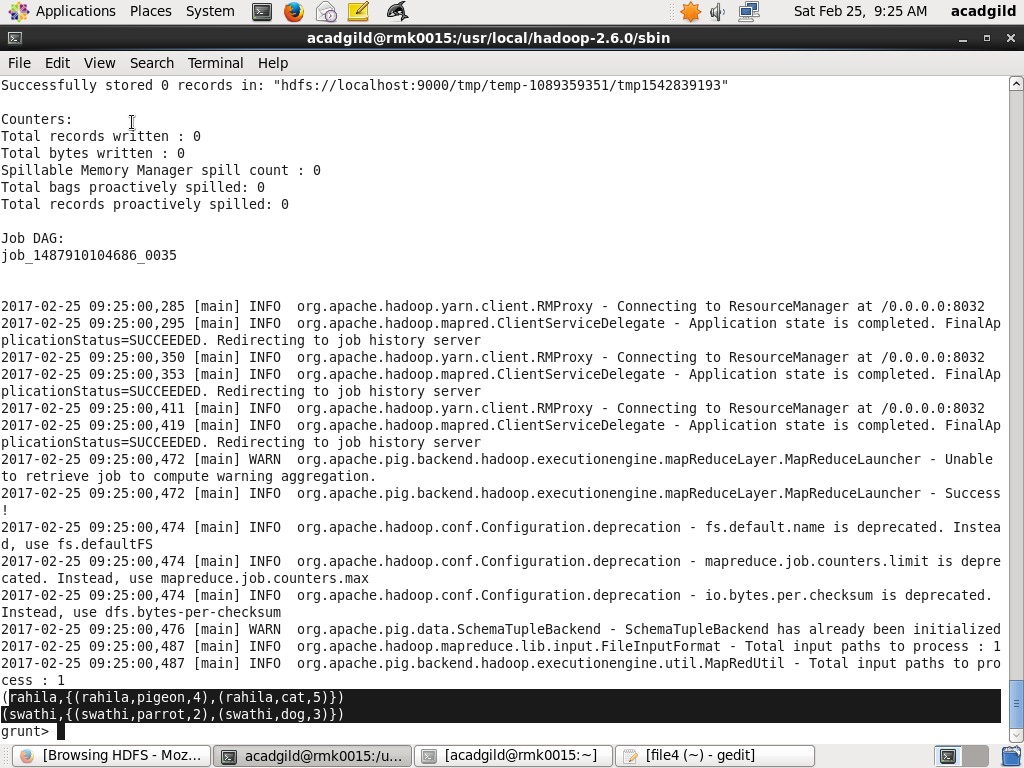


Dump the file4\_loading:

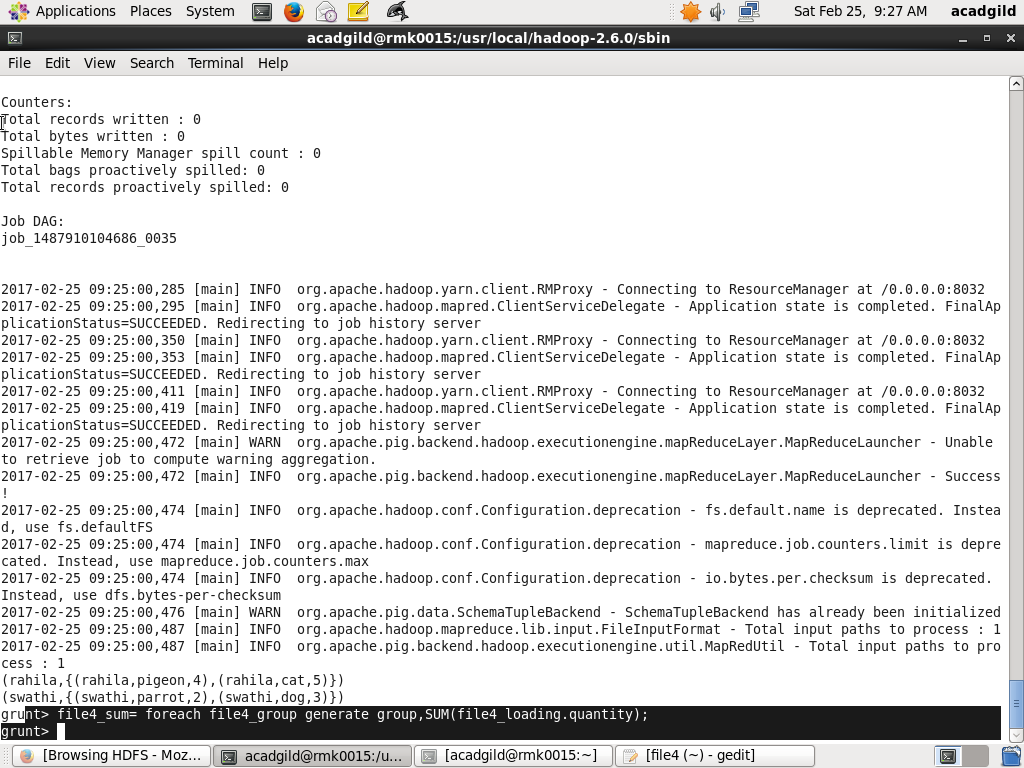


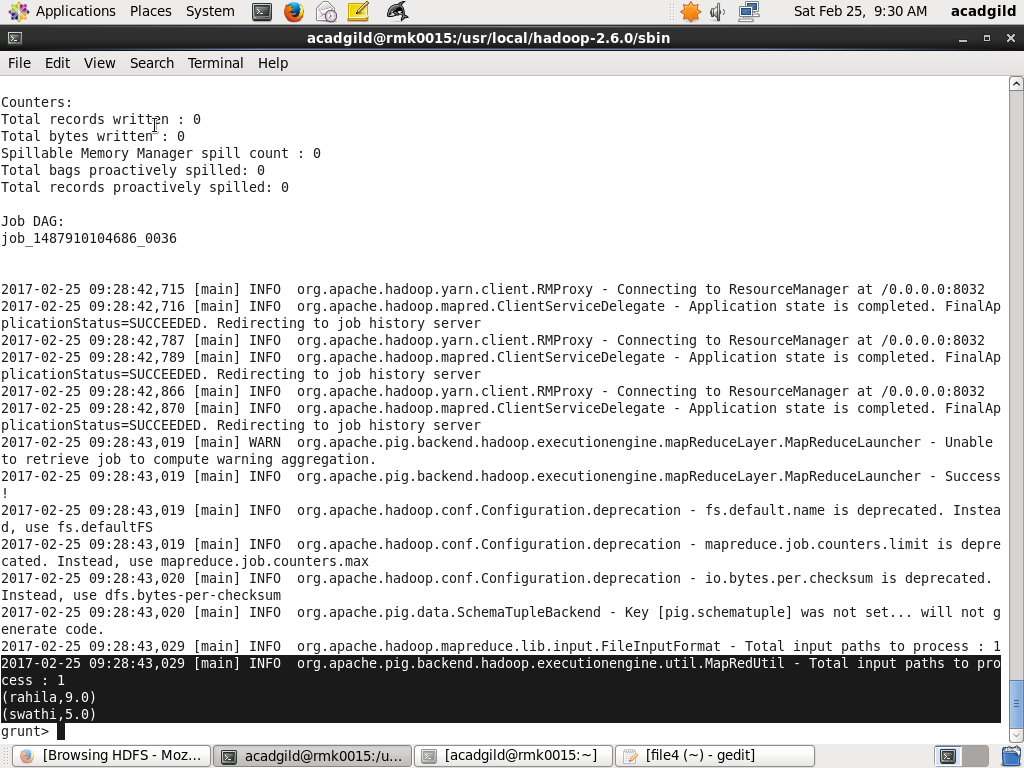
Group the file4\_loading file by name:





Sum the quantity and give the name as file4\_sum:





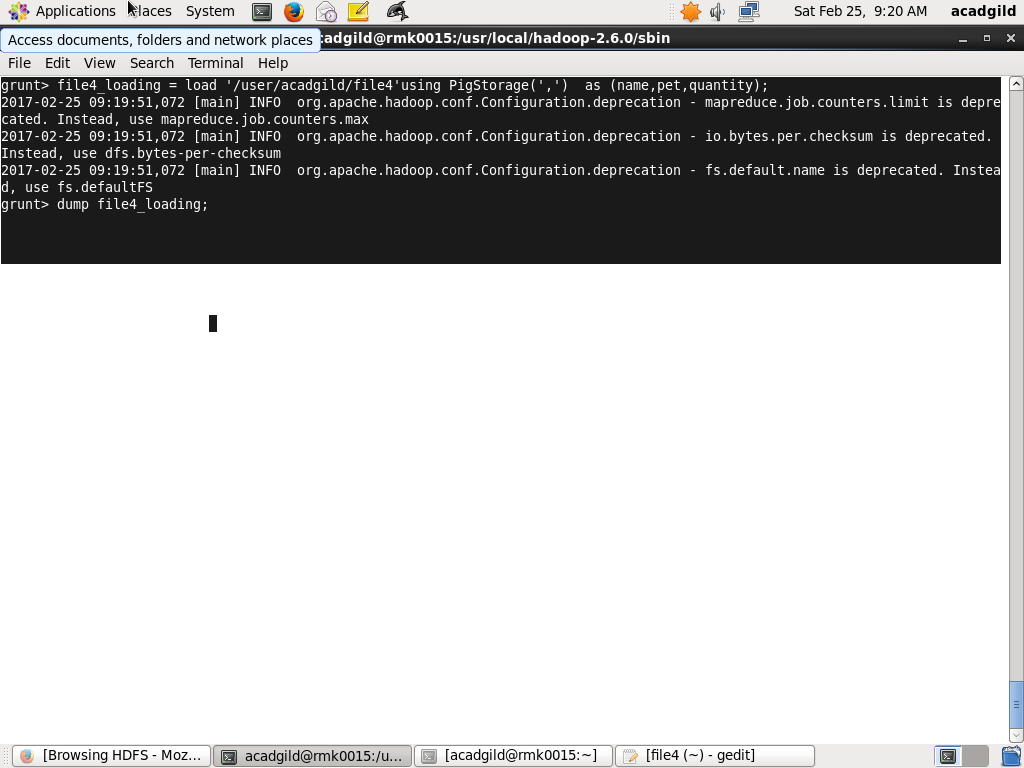
**MIN:**

Computes the minimum of the numeric values or chararrays in a single-column bag.

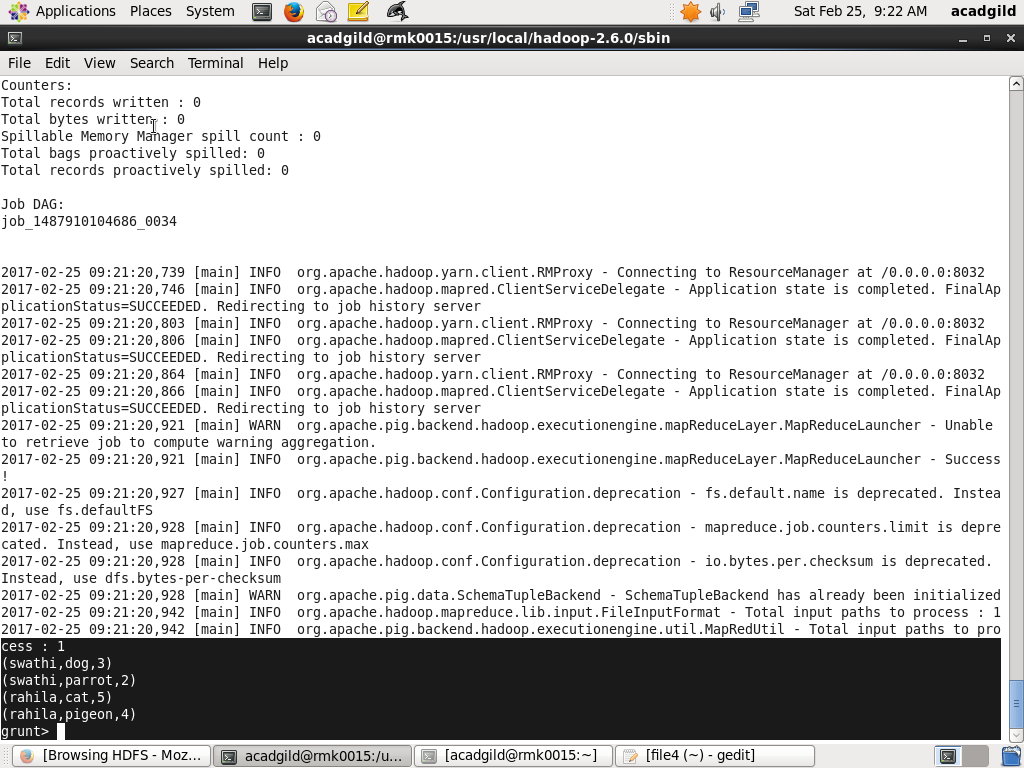
Syntax:

MIN(expression)

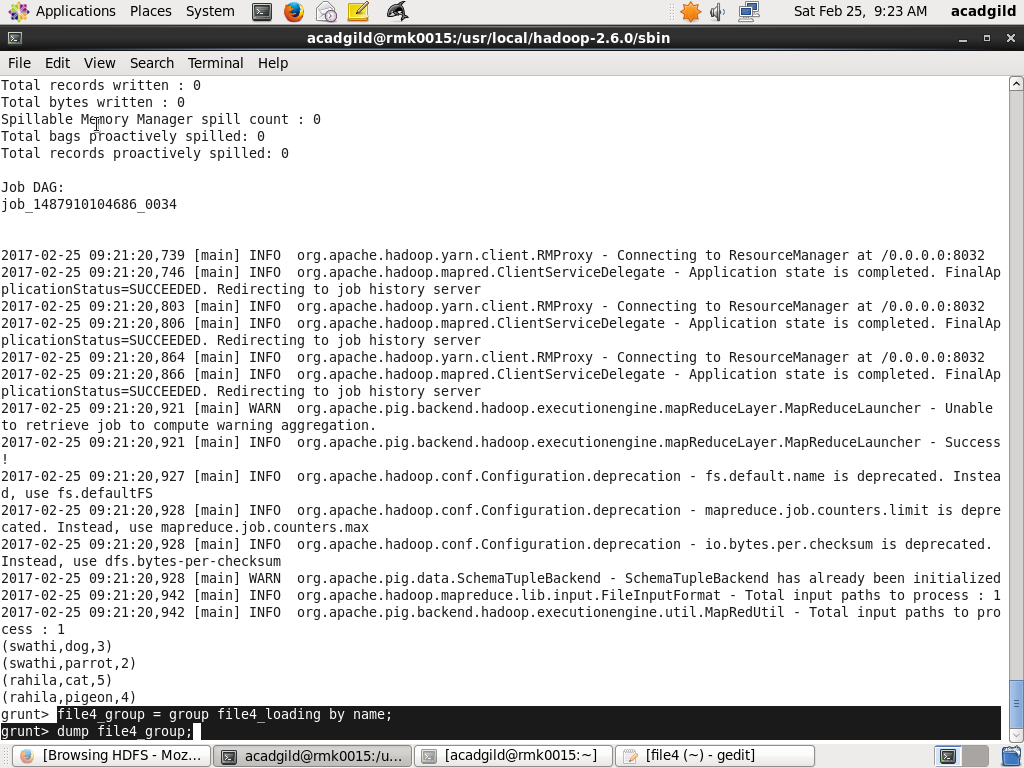
Load the file4 into file4\_loading:

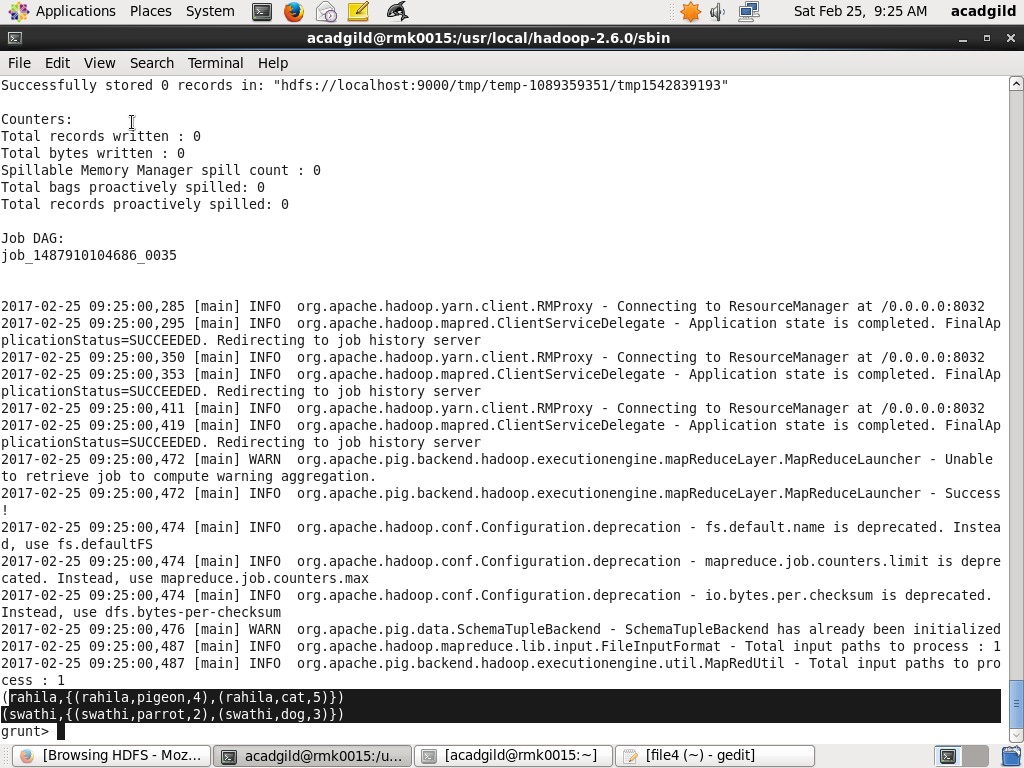


Dump the file4\_loading:



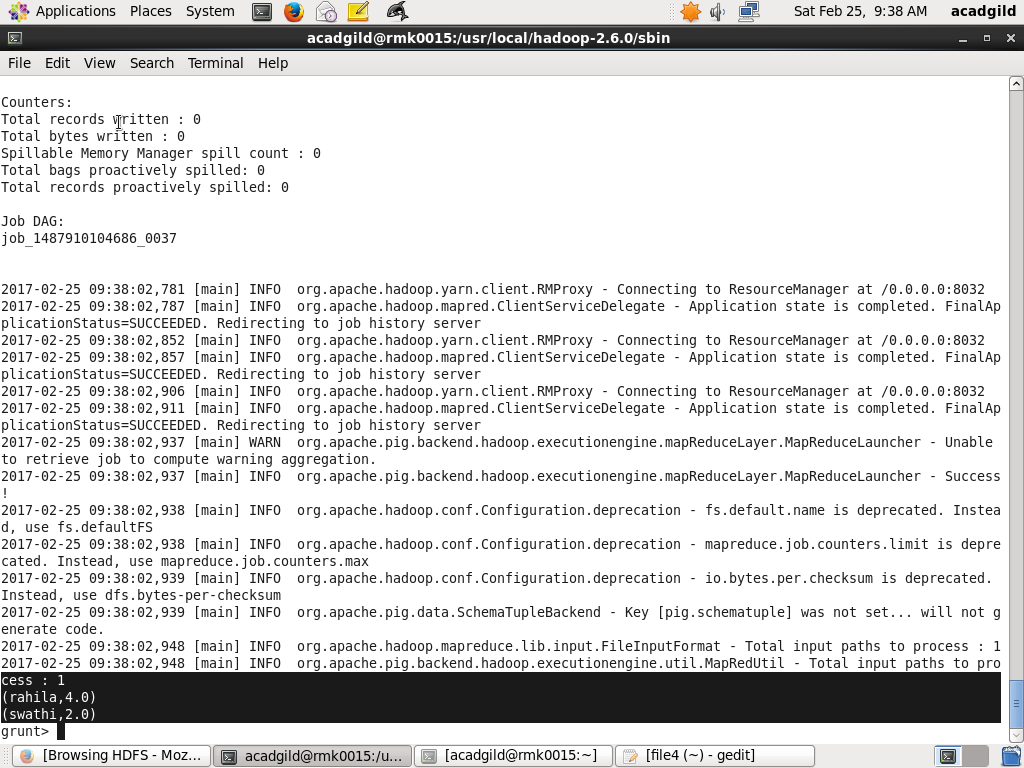
Group the file4\_loading file by name:





Min the quantity and give the name as file4\_min:





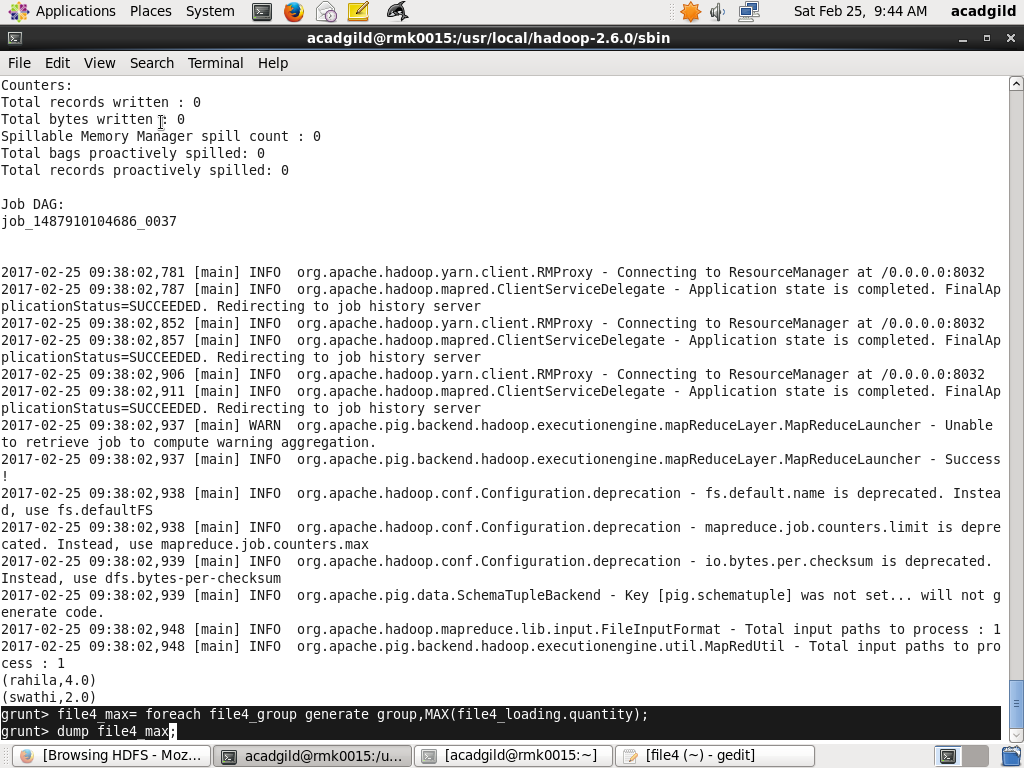
**MAX:**

Computes the maximum of the numeric values or chararrays in a single-column bag.

Syntax:

MAX(expression)

Apply the maximum comment on file4\_group:



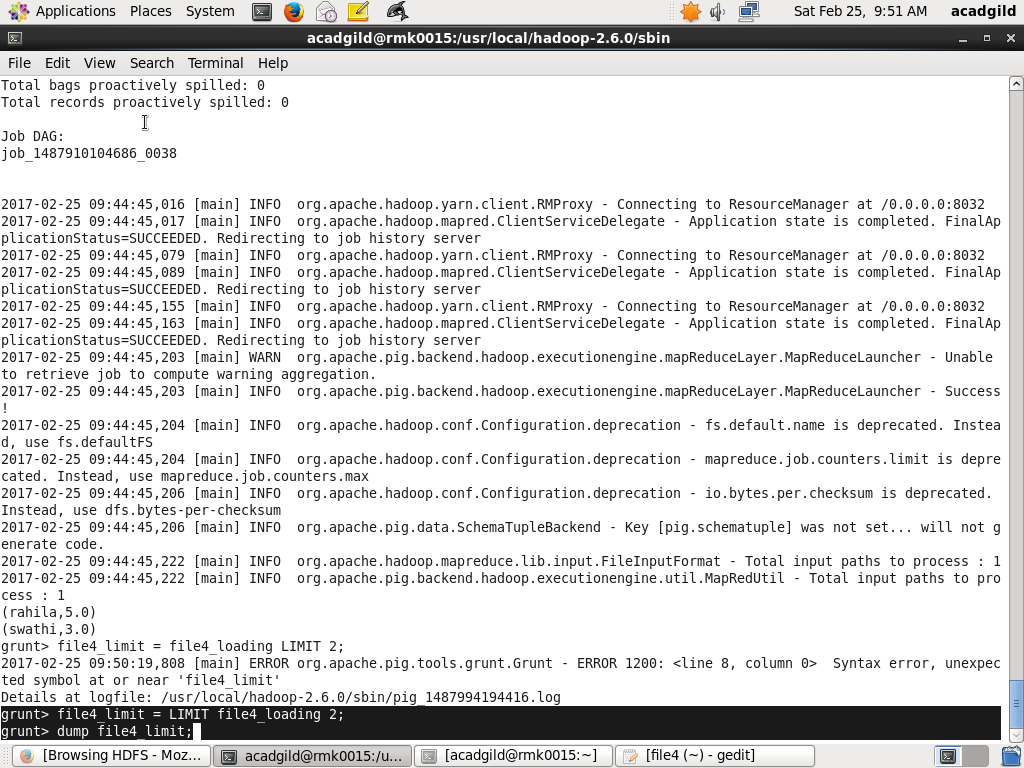


**LIMIT:**

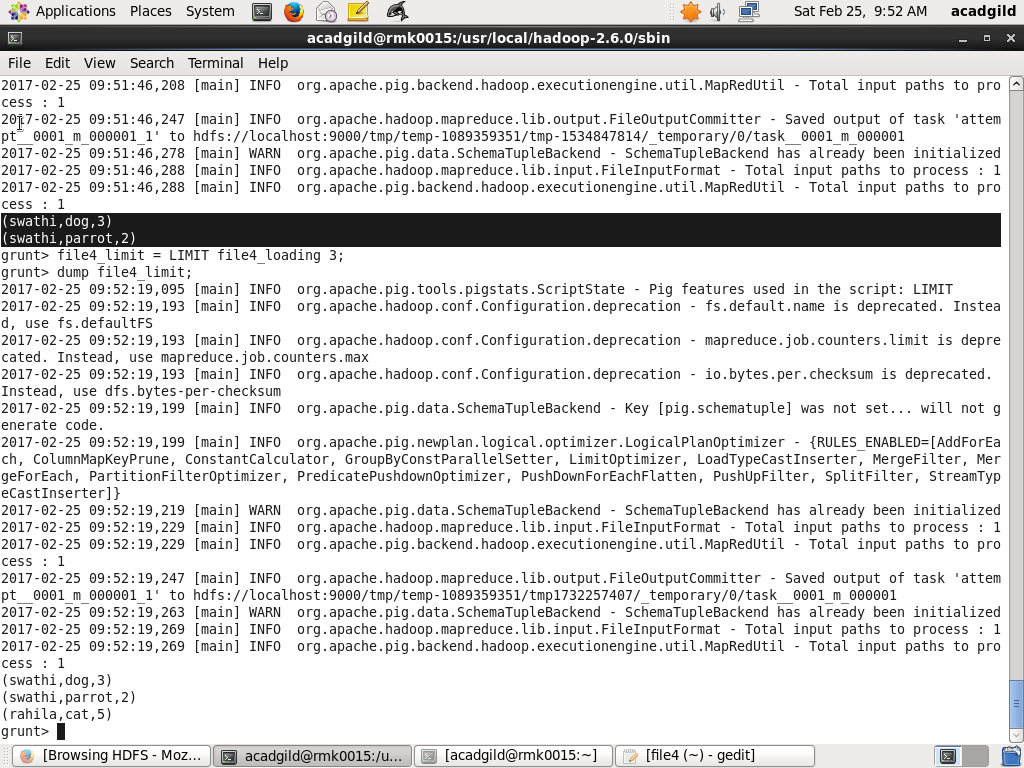
Limits the number of output tuples.

Syntax:

alias = LIMIT alias  n;



The LIMIT operator is used to output the first three tuples.



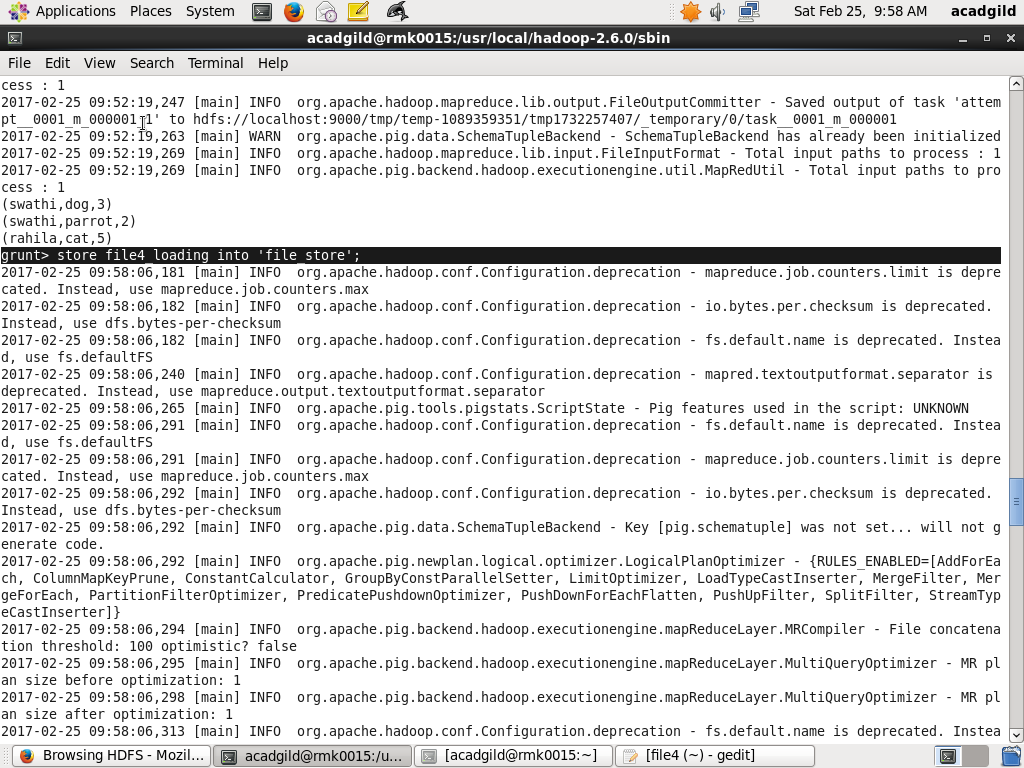
**STORE:**

Stores or saves results to the file system.

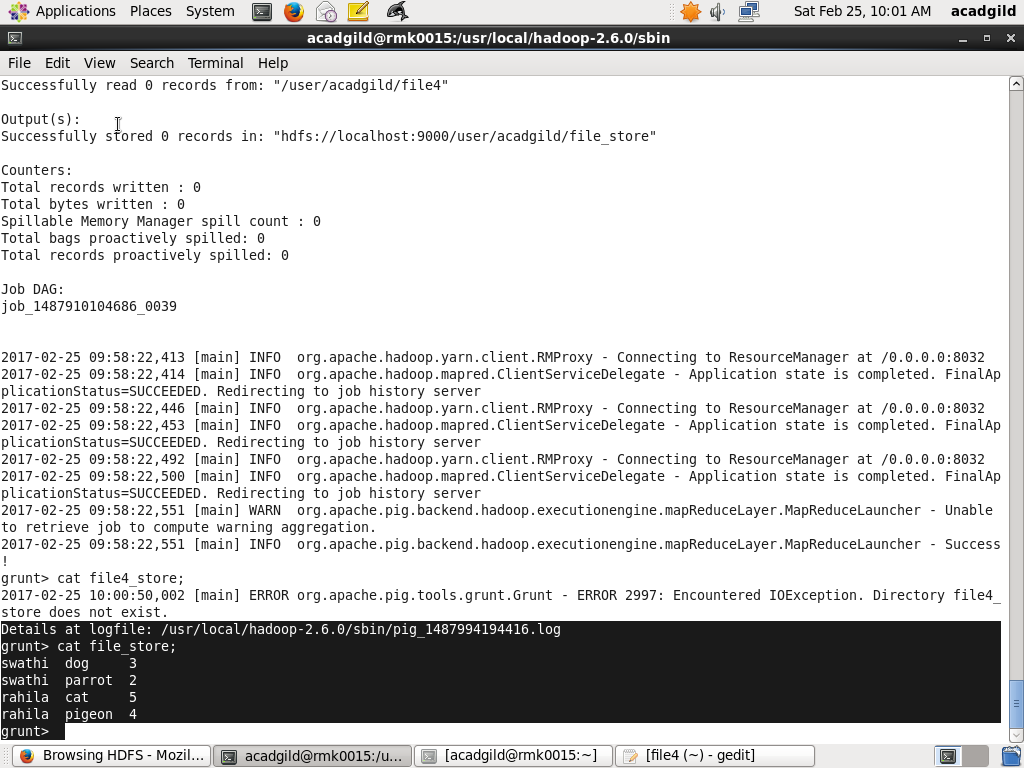
Syntax:

STORE alias INTO 'directory' [USING function];

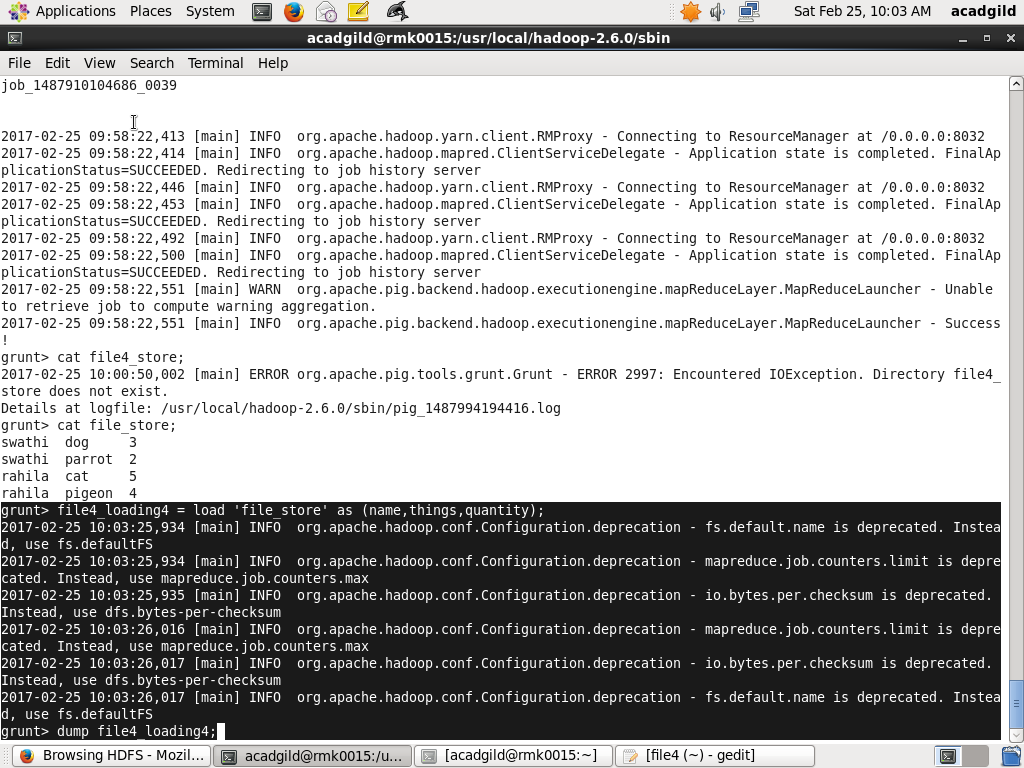
Store the file4\_loading into file\_store:

****

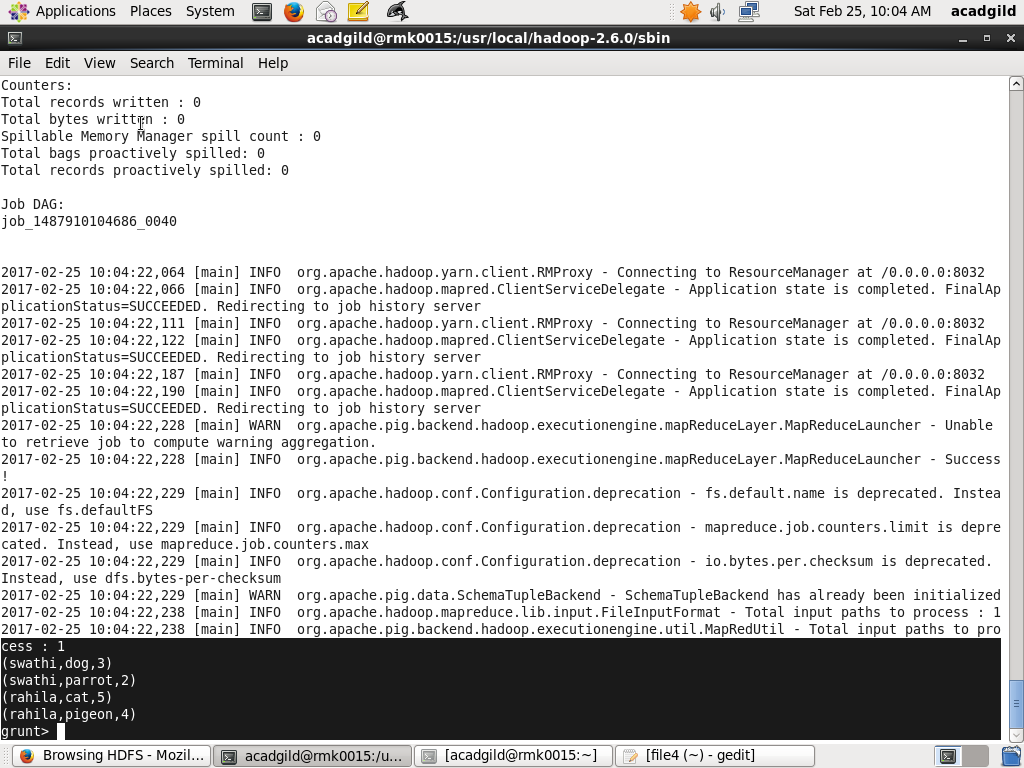
Using cat command the data is displayed:

****

File\_store is loaded into file4\_loading4 file:

****

Using dump statement the file is displayed:

****

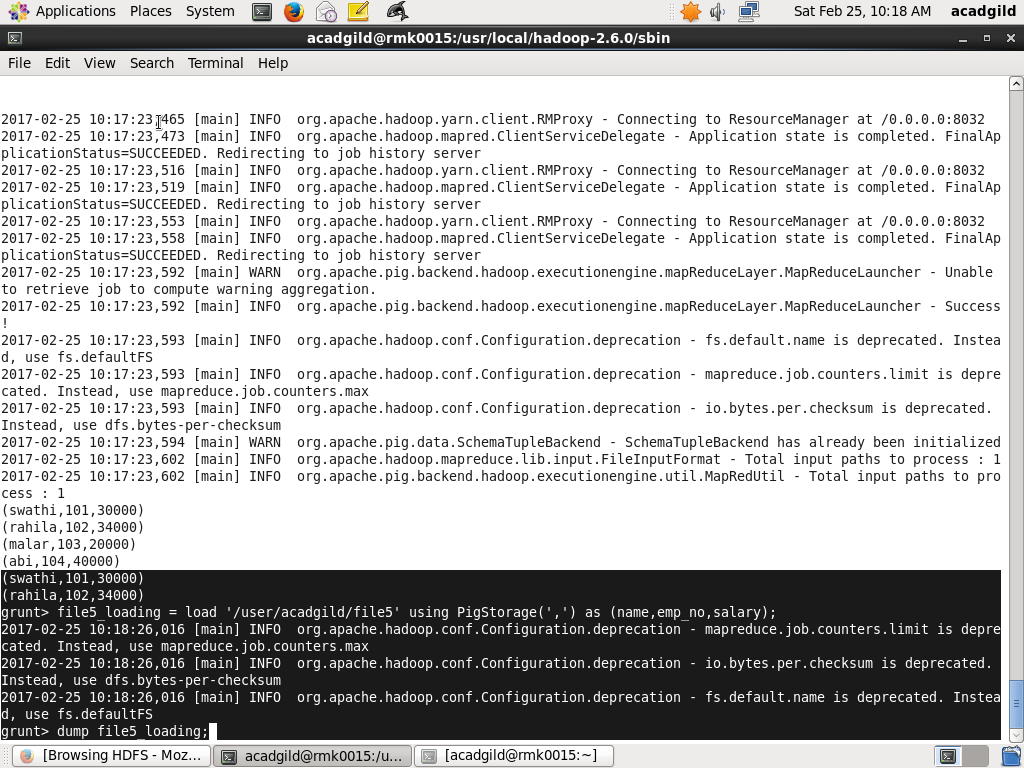
**DISTINCT:**

Removes duplicate tuples in a relation.

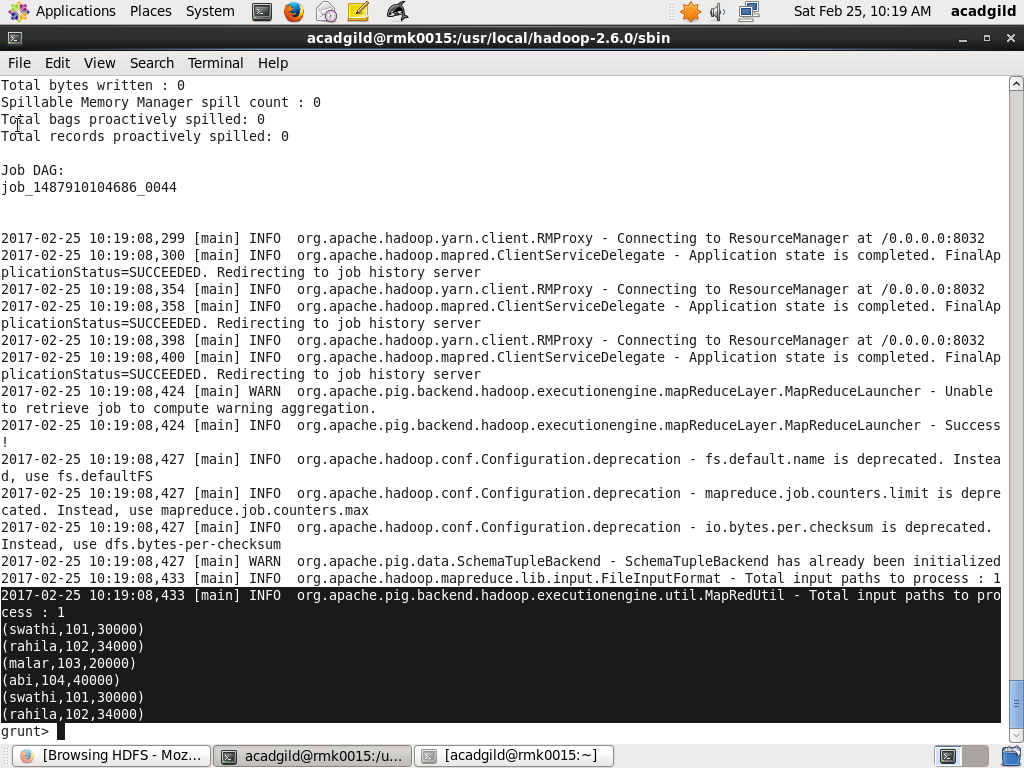
Syntax:

alias = DISTINCT alias [PARALLEL n];

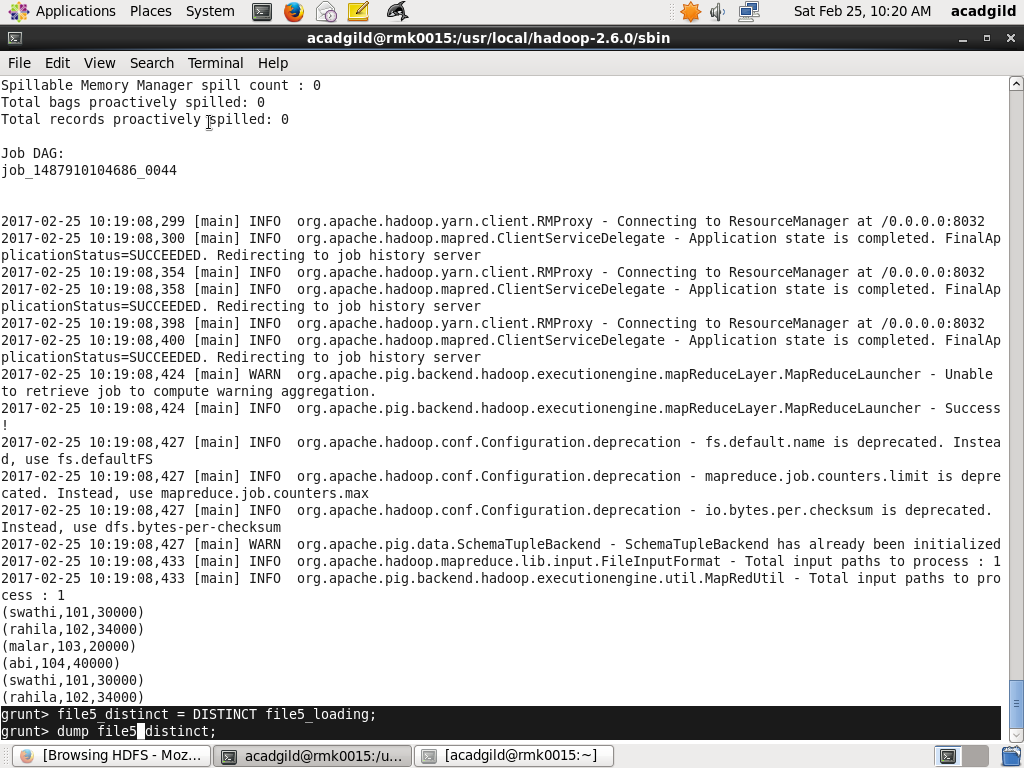
Create the file5 with duplicates data and it is loaded into file5\_loading:

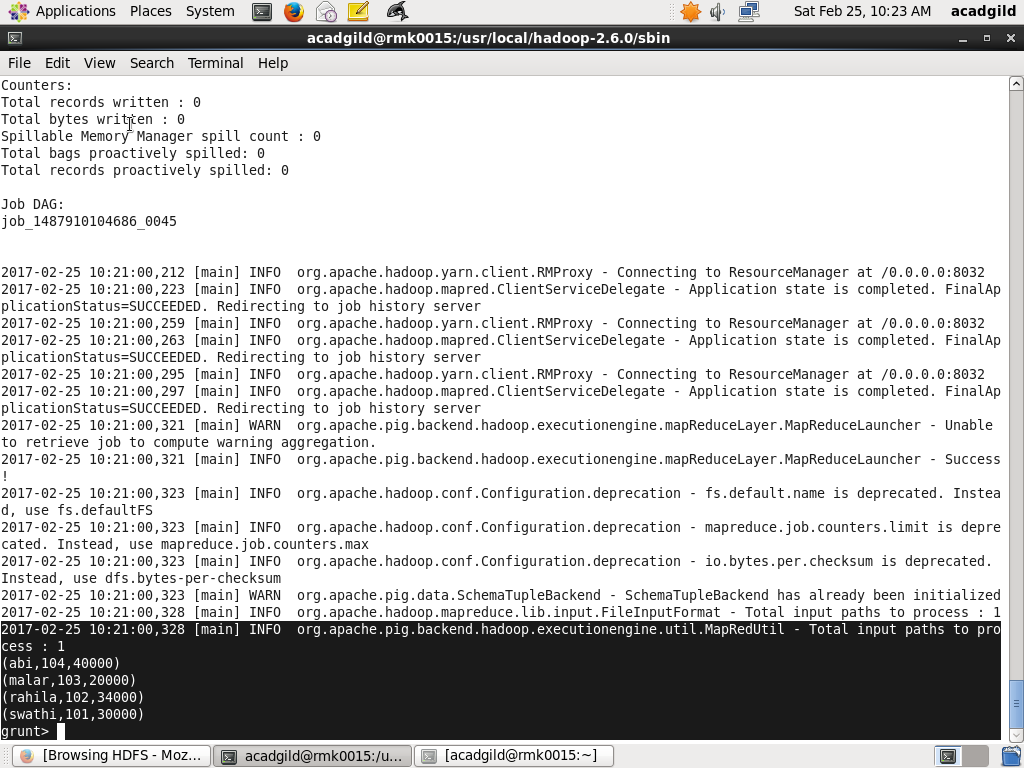


Display the content usinmg dump statement:



Apply the distinct command into file5\_loading and give the name as file5\_distinct:





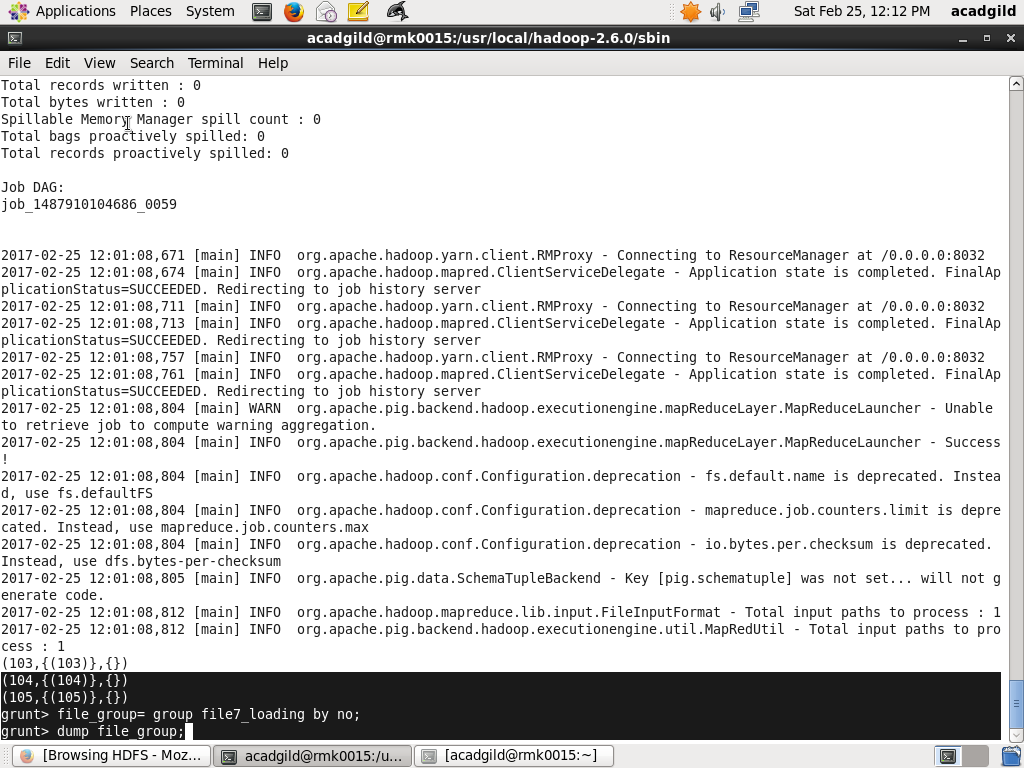
**FLATTEN:**

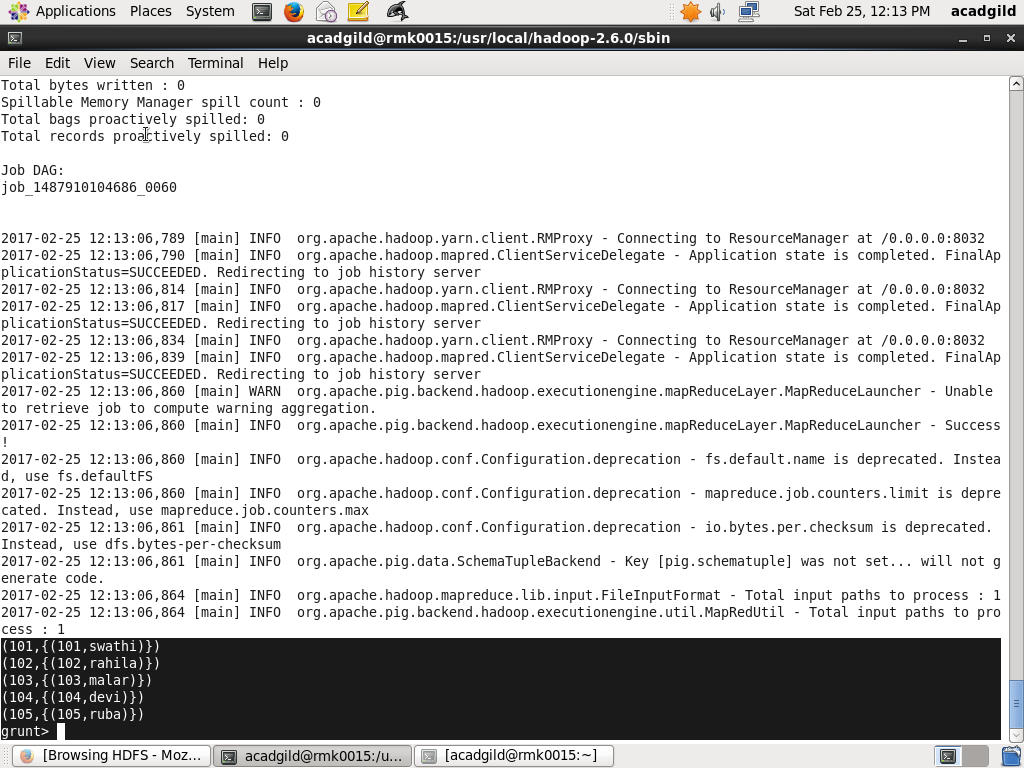
The FLATTEN operator looks like a UDF syntactically, but it is actually an operator that changes the structure of tuples and bags in a way that a UDF cannot. Flatten un-nests tuples as well as bags. The idea is the same, but the operation and result is different for each type of structure.

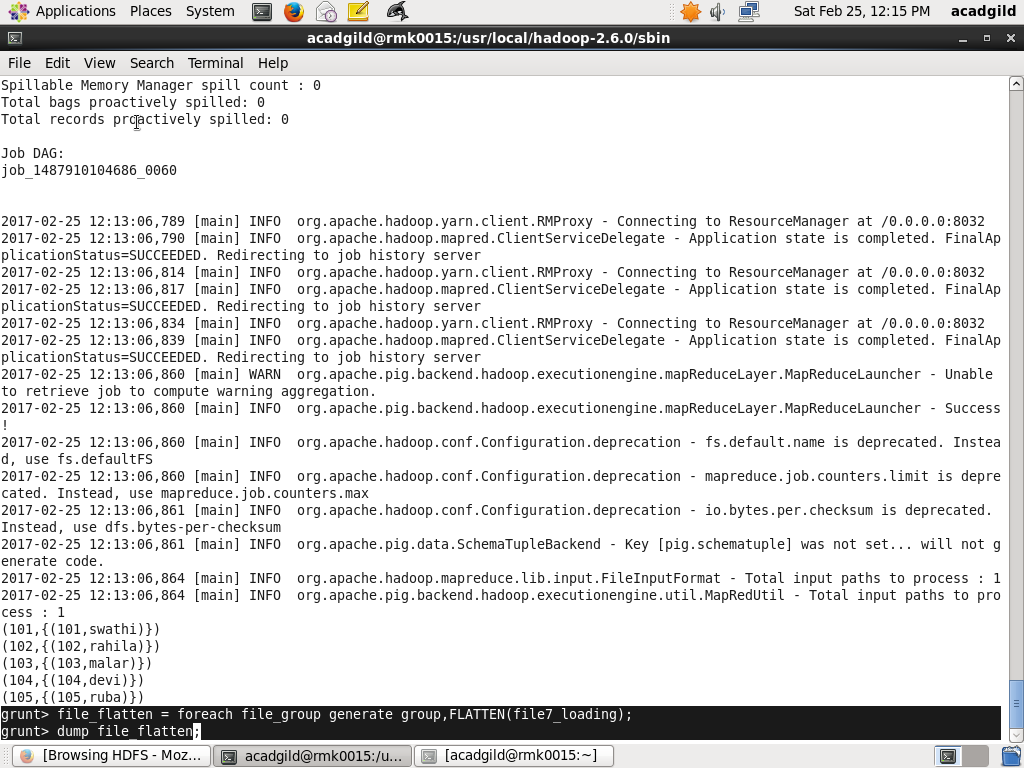
Syntax:

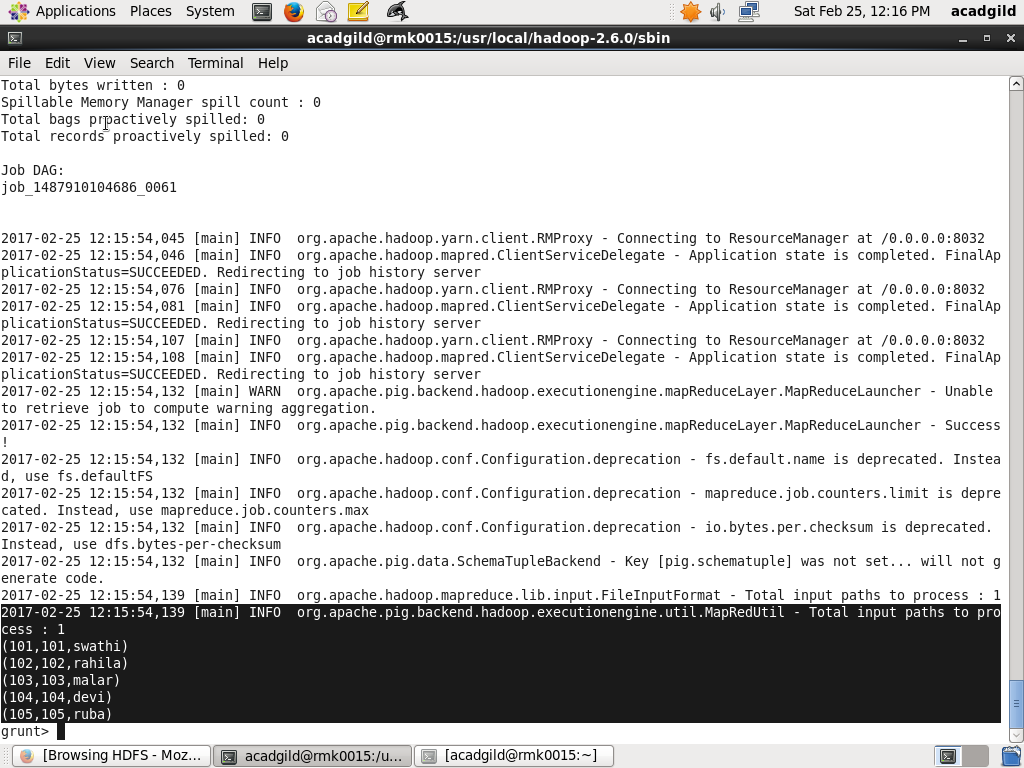
{(data\_type) |  (tuple(data\_type))  | (bag{tuple(data\_type)}) | (map[]) } field

Create the file which is file7. It is grouped based on the number which is no and then apply the flatten statement to avoid the nested bags of the data:









**IsEmpty:**

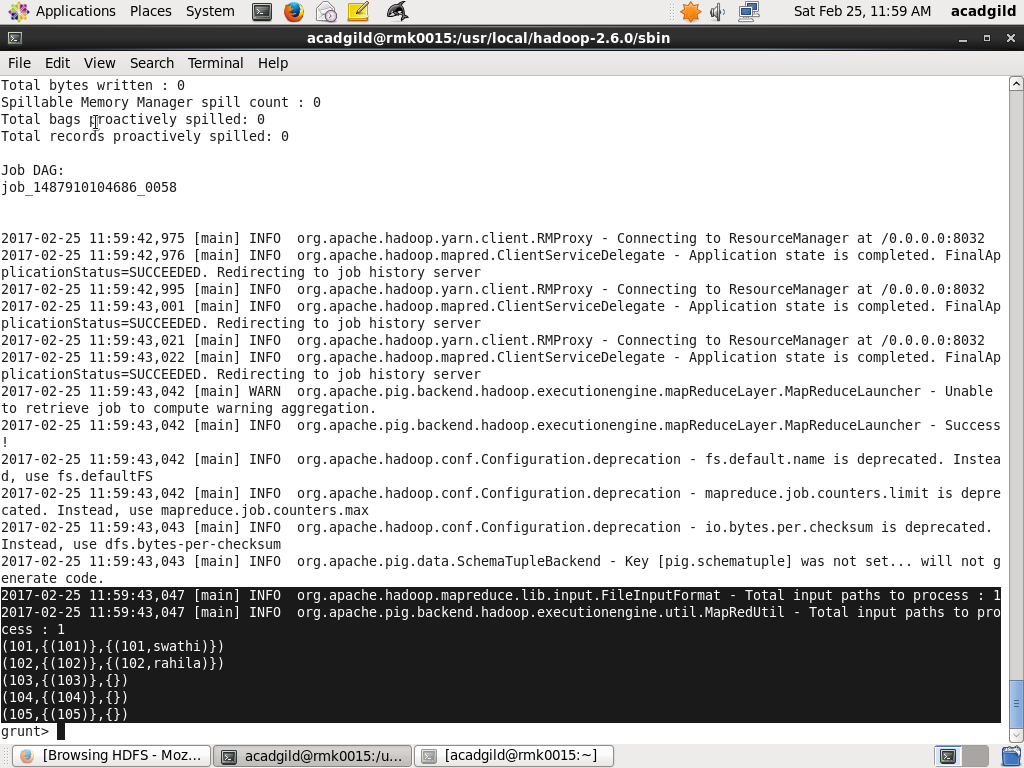
Checks if a bag or map is empty.

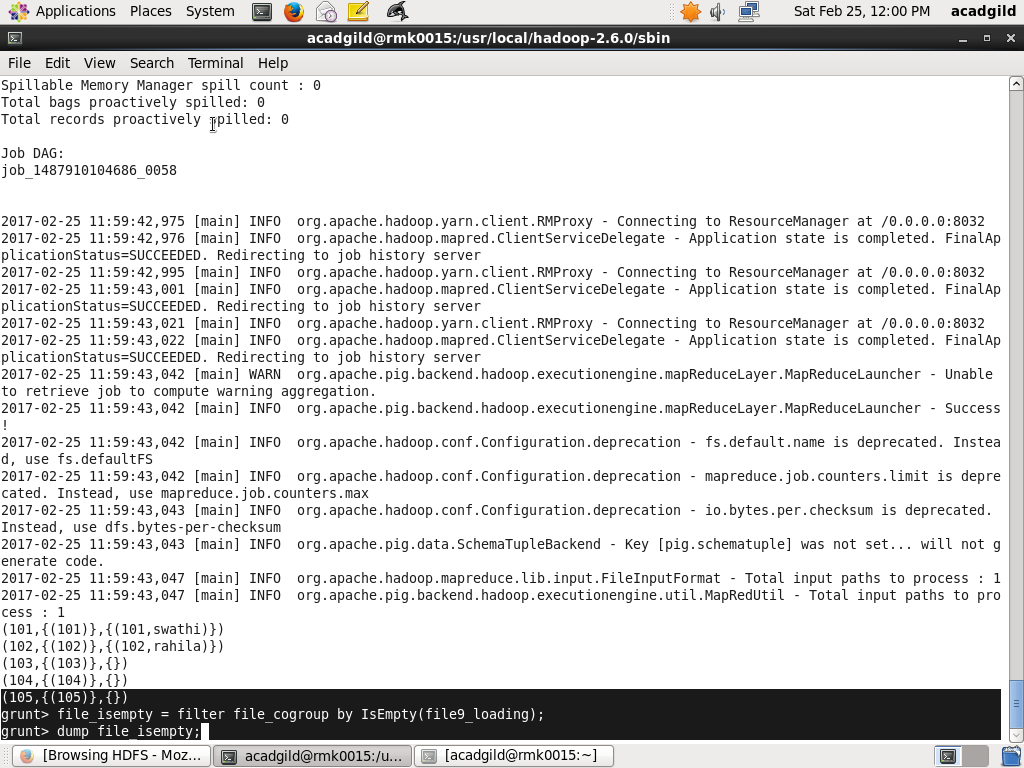
Syntax:

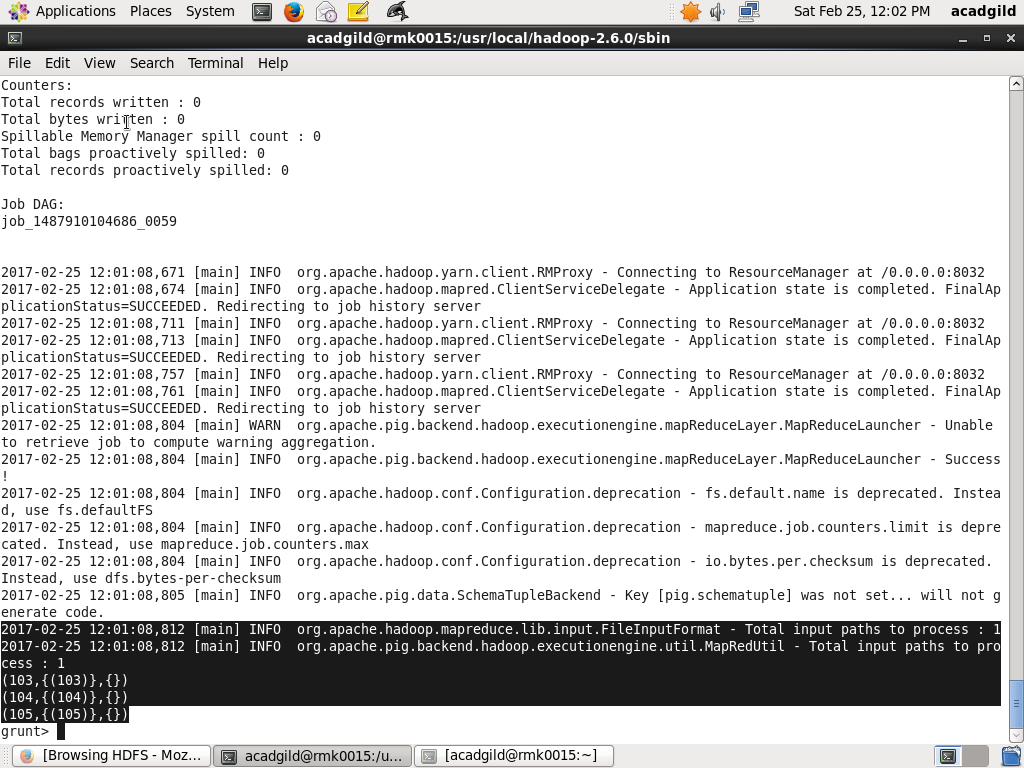
IsEmpty(expression)

Create the 2 files which are file6 and file9. Both the files are loaded into file6\_loading and file9\_loading seperately. Apply the cogroup commnd to group the files by number which is no. Finally apply the IsEmpty command for only keep those number’s which there is no name:

****

****

****

****