

# Testing and Diagnostics

## Table of contents

1 Diagnostic Operators.....	2
2 Pig Scripts and MapReduce Job IDs.....	9
3 Pig Statistics.....	9
4 Pig Progress Notification Listener.....	12
5 PigUnit.....	12

## 1 Diagnostic Operators

### 1.1 DESCRIBE

Returns the schema of a relation.

#### 1.1.1 Syntax

DESCRIBE alias;
-----------------

#### 1.1.2 Terms

alias	The name of a relation.
-------	-------------------------

#### 1.1.3 Usage

Use the DESCRIBE operator to view the schema of a relation. You can view outer relations as well as relations defined in a nested FOREACH statement.

#### 1.1.4 Example

In this example a schema is specified using the AS clause. If all data conforms to the schema, Pig will use the assigned types.

```
A = LOAD 'student' AS (name:chararray, age:int, gpa:float);
B = FILTER A BY name matches 'J.+';
C = GROUP B BY name;
D = FOREACH B GENERATE COUNT(B.age);

DESCRIBE A;
A: {group, B: (name: chararray,age: int,gpa: float)}

DESCRIBE B;
B: {group, B: (name: chararray,age: int,gpa: float)}

DESCRIBE C;
C: {group, chararray,B: (name: chararray,age: int,gpa: float)}

DESCRIBE D;
D: {long}
```

In this example no schema is specified. All fields default to type bytearray or long (see Data Types).

```
a = LOAD 'student';
```

```

b = FILTER a BY $0 matches 'J.+';

c = GROUP b BY $0;

d = FOREACH c GENERATE COUNT(b.$1);

DESCRIBE a;
Schema for a unknown.

DESCRIBE b;
2008-12-05 01:17:15,316 [main] WARN org.apache.pig.PigServer - bytearray is implicitly
cast to chararray under LORegexp Operator
Schema for b unknown.

DESCRIBE c;
2008-12-05 01:17:23,343 [main] WARN org.apache.pig.PigServer - bytearray is implicitly
cast to chararray under LORegexp Operator
c: {group: bytearray,b: {null}}

DESCRIBE d;
2008-12-05 03:04:30,076 [main] WARN org.apache.pig.PigServer - bytearray is implicitly
cast to chararray under LORegexp Operator
d: {long}

```

This example shows how to view the schema of a nested relation using the :: operator.

```

A = LOAD 'studentab10k' AS (name, age, gpa);
B = GROUP A BY name;
C = FOREACH B {
    D = DISTINCT A.age;
    GENERATE COUNT(D), group;}

DESCRIBE C::D;
D: {age: bytearray}

```

## 1.2 DUMP

Dumps or displays results to screen.

### 1.2.1 Syntax

DUMP alias;
-------------

### 1.2.2 Terms

alias	The name of a relation.
-------	-------------------------

### 1.2.3 Usage

Use the DUMP operator to run (execute) Pig Latin statements and display the results to your screen. DUMP is meant for interactive mode; statements are executed immediately and the

results are not saved (persisted). You can use DUMP as a debugging device to make sure that the results you are expecting are actually generated.

Note that production scripts **SHOULD NOT** use DUMP as it will disable multi-query optimizations and is likely to slow down execution (see [Store vs. Dump](#)).

### 1.2.4 Example

In this example a dump is performed after each statement.

```
A = LOAD 'student' AS (name:chararray, age:int, gpa:float);

DUMP A;
(John,18,4.0F)
(Mary,19,3.7F)
(Bill,20,3.9F)
(Joe,22,3.8F)
(Jill,20,4.0F)

B = FILTER A BY name matches 'J.+';

DUMP B;
(John,18,4.0F)
(Joe,22,3.8F)
(Jill,20,4.0F)
```

## 1.3 EXPLAIN

Displays execution plans.

### 1.3.1 Syntax

```
EXPLAIN [--script pigscript] [--out path] [--brief] [--dot] [--param param_name = param_value] [--param_file file_name] alias;
```

### 1.3.2 Terms

--script	Use to specify a Pig script.
--out	Use to specify the output path (directory). Will generate a logical_plan[.txt .dot], physical_plan[.text .dot], exec_plan[.text .dot] file in the specified path. Default (no path specified): Stdout
--brief	Does not expand nested plans (presenting a smaller graph for overview).

<code>-dot</code>	Text mode (default): multiple output (split) will be broken out in sections.  Dot mode: outputs a format that can be passed to the dot utility for graphical display – will generate a directed-acyclic-graph (DAG) of the plans in any supported format (.gif, .jpg ...).
<code>-param param_name = param_value</code>	See <a href="#">Parameter Substitution</a> .
<code>-param_file file_name</code>	See <a href="#">Parameter Substitution</a> .
<code>alias</code>	The name of a relation.

### 1.3.3 Usage

Use the EXPLAIN operator to review the logical, physical, and map reduce execution plans that are used to compute the specified relationship.

If no script is given:

- The logical plan shows a pipeline of operators to be executed to build the relation. Type checking and backend-independent optimizations (such as applying filters early on) also apply.
- The physical plan shows how the logical operators are translated to backend-specific physical operators. Some backend optimizations also apply.
- The mapreduce plan shows how the physical operators are grouped into map reduce jobs.

If a script without an alias is specified, it will output the entire execution graph (logical, physical, or map reduce).

If a script with a alias is specified, it will output the plan for the given alias.

### 1.3.4 Example

In this example the EXPLAIN operator produces all three plans. (Note that only a portion of the output is shown in this example.)

```
A = LOAD 'student' AS (name:chararray, age:int, gpa:float);
B = GROUP A BY name;
C = FOREACH B GENERATE COUNT(A.age);

EXPLAIN C;
-----
Logical Plan:
-----
Store xxx-Fri Dec 05 19:42:29 UTC 2008-23 Schema: {long} Type: Unknown
|
```

```

|---ForEach xxx-Fri Dec 05 19:42:29 UTC 2008-15 Schema: {long} Type: bag
etc ...

-----
Physical Plan:
-----
Store(fakefile:org.apache.pig.builtin.PigStorage) - xxx-Fri Dec 05 19:42:29 UTC 2008-40
|
|---New For Each(false)[bag] - xxx-Fri Dec 05 19:42:29 UTC 2008-39
|   |
|   | POUserFunc(org.apache.pig.builtin.COUNT)[long] - xxx-Fri Dec 05
etc ...

-----
| Map Reduce Plan
-----
MapReduce node xxx-Fri Dec 05 19:42:29 UTC 2008-41
Map Plan
Local Rearrange[tuple]{chararray}(false) - xxx-Fri Dec 05 19:42:29 UTC 2008-34
|   |
|   | Project[chararray][0] - xxx-Fri Dec 05 19:42:29 UTC 2008-35
etc ...

```

## 1.4 ILLUSTRATE

Displays a step-by-step execution of a sequence of statements.

### 1.4.1 Syntax

```
ILLUSTRATE {alias | -script scriptfile};
```

### 1.4.2 Terms

alias	The name of a relation.
-script scriptfile	<p>The script keyword followed by the name of a Pig script (for example, myscript.pig).</p> <p>The script file should not contain an ILLUSTRATE statement.</p>

### 1.4.3 Usage

Use the ILLUSTRATE operator to review how data is transformed through a sequence of Pig Latin statements. ILLUSTRATE allows you to test your programs on small datasets and get faster turnaround times.

ILLUSTRATE is based on an example generator (see [Generating Example Data for Dataflow Programs](#)). The algorithm works by retrieving a small sample of the input data and then propagating this data through the pipeline. However, some operators, such as JOIN and

**FILTER**, can eliminate tuples from the data - and this could result in no data following through the pipeline. To address this issue, the algorithm will automatically generate example data, in near real-time. Thus, you might see data propagating through the pipeline that was not found in the original input data, but this data changes nothing and ensures that you will be able to examine the semantics of your Pig Latin statements.

As shown in the examples below, you can use **ILLUSTRATE** to review a relation or an entire Pig script.

#### 1.4.4 Example - Relation

This example demonstrates how to use **ILLUSTRATE** with a relation. Note that the **LOAD** statement must include a schema (the **AS** clause).

```
grunt> visits = LOAD 'visits.txt' AS (user:chararray, url:chararray, timestamp:chararray);
grunt> DUMP visits;
```

```
(Amy,yahoo.com,19990421)
(Fred,harvard.edu,19991104)
(Amy,cnn.com,20070218)
(Frank,nba.com,20070305)
(Fred,berkeley.edu,20071204)
(Fred,stanford.edu,20071206)
```

```
grunt> recent_visits = FILTER visits BY timestamp >= '20071201';
grunt> user_visits = GROUP recent_visits BY user;
grunt> num_user_visits = FOREACH user_visits GENERATE group, COUNT(recent_visits);
grunt> DUMP num_user_visits;
```

```
(Fred,2)
```

```
grunt> ILLUSTRATE num_user_visits;
```

visits	user: chararray	url: chararray	timestamp: chararray
	Fred	berkeley.edu	20071204
	Fred	stanford.edu	20071206
	Frank	nba.com	20070305

  

recent_visits	user: chararray	url: chararray	timestamp: chararray
	Fred	berkeley.edu	20071204
	Fred	stanford.edu	20071206

  

user_visits	group: chararray	recent_visits: bag({user: chararray,url: chararray,timestamp: chararray})
	Fred	{(Fred, berkeley.edu, 20071204), (Fred, stanford.edu, 20071206)}

  

num_user_visits	group: chararray	long
	Fred	2

	Fred	2
--	------	---

### 1.4.5 Example - Script

This example demonstrates how to use ILLUSTRATE with a Pig script. Note that the script itself should not contain an ILLUSTRATE statement.

```
grunt> cat visits.txt
Amy    yahoo.com    19990421
Fred   harvard.edu   19991104
Amy    cnn.com 20070218
Frank  nba.com 20070305
Fred   berkeley.edu  20071204
Fred   stanford.edu   20071206
```

```
grunt> cat visits.pig
visits = LOAD 'visits.txt' AS (user, url, timestamp);
recent_visits = FILTER visits BY timestamp >= '20071201';
historical_visits = FILTER visits BY timestamp <= '20000101';
DUMP recent_visits;
DUMP historical_visits;
STORE recent_visits INTO 'recent';
STORE historical_visits INTO 'historical';
```

```
grunt> exec visits.pig
```

```
(Fred,berkeley.edu,20071204)
(Fred,stanford.edu,20071206)
```

```
(Amy,yahoo.com,19990421)
(Fred,harvard.edu,19991104)
```

```
grunt> illustrate -script visits.pig
```

visits	user: bytearray	url: bytearray	timestamp: bytearray
	Amy	yahoo.com	19990421
	Fred	stanford.edu	20071206

recent_visits	user: bytearray	url: bytearray	timestamp: bytearray
	Fred	stanford.edu	20071206

Store : recent_visits	user: bytearray	url: bytearray	timestamp: bytearray
	Fred	stanford.edu	20071206

historical_visits	user: bytearray	url: bytearray	timestamp: bytearray
	Amy	yahoo.com	19990421



```

-----
| Store : historical_visits      | user: bytearray | url: bytearray | timestamp: bytearray |
|                               | Amy            | yahoo.com      | 19990421             |
-----

```

## 2 Pig Scripts and MapReduce Job IDs

Complex Pig scripts often generate many MapReduce jobs. To help you debug a script, Pig prints a summary of the execution that shows which relations (aliases) are mapped to each MapReduce job.

```

JobId Maps Reduces MaxMapTime MinMapTime AvgMapTime MaxReduceTime
MinReduceTime AvgReduceTime Alias Feature Outputs
job_201004271216_12712 1 1 3 3 3 12 12 12 B,C GROUP_BY,COMBINER
job_201004271216_12713 1 1 3 3 3 12 12 12 D SAMPLER
job_201004271216_12714 1 1 3 3 3 12 12 12 D ORDER_BY,COMBINER
hdfs://mymachine.com:9020/tmp/temp743703298/tmp-2019944040,

```

## 3 Pig Statistics

Pig Statistics is a framework for collecting and storing script-level statistics for Pig Latin. Characteristics of Pig Latin scripts and the resulting MapReduce jobs are collected while the script is executed. These statistics are then available for Pig users and tools using Pig (such as Oozie) to retrieve after the job is done.

The new Pig statistics and the existing Hadoop statistics can also be accessed via the Hadoop job history file (and job xml file). Piggybank has a HadoopJobHistoryLoader which acts as an example of using Pig itself to query these statistics (the loader can be used as a reference implementation but is NOT supported for production use).

### 3.1 Java API

Several new public classes make it easier for external tools such as Oozie to integrate with Pig statistics.

The Pig statistics are available here: <http://pig.apache.org/docs/r0.9.0/api/>

The stats classes are in the package: org.apache.pig.tools.pigstats

- PigStats
- JobStats
- OutputStats
- InputStats

The `PigRunner` class mimics the behavior of the `Main` class but gives users a statistics object back. Optionally, you can call the API with an implementation of progress listener which will be invoked by Pig runtime during the execution.

```
package org.apache.pig;

public abstract class PigRunner {
    public static PigStats run(String[] args, PigProgressNotificationListener listener)
    }

public interface PigProgressNotificationListener extends java.util.EventListener {
    // just before the launch of MR jobs for the script
    public void LaunchStartedNotification(int numJobsToLaunch);
    // number of jobs submitted in a batch
    public void jobsSubmittedNotification(int numJobsSubmitted);
    // a job is started
    public void jobStartedNotification(String assignedJobId);
    // a job is completed successfully
    public void jobFinishedNotification(JobStats jobStats);
    // a job is failed
    public void jobFailedNotification(JobStats jobStats);
    // a user output is completed successfully
    public void outputCompletedNotification(OutputStats outputStats);
    // updates the progress as percentage
    public void progressUpdatedNotification(int progress);
    // the script execution is done
    public void launchCompletedNotification(int numJobsSucceeded);
}
```

### 3.2 Job XML

The following entries are included in job conf:

Pig Statistic	Description
<code>pig.script.id</code>	The UUID for the script. All jobs spawned by the script have the same script ID.
<code>pig.script</code>	The base64 encoded script text.
<code>pig.command.line</code>	The command line used to invoke the script.
<code>pig.hadoop.version</code>	The Hadoop version installed.
<code>pig.version</code>	The Pig version used.
<code>pig.input.dirs</code>	A comma-separated list of input directories for the job.
<code>pig.map.output.dirs</code>	A comma-separated list of output directories in the map phase of the job.

pig.reduce.output.dirs	A comma-separated list of output directories in the reduce phase of the job.
pig.parent.jobid	A comma-separated list of parent job ids.
pig.script.features	A list of Pig features used in the script.
pig.job.feature	A list of Pig features used in the job.
pig.alias	The alias associated with the job.

### 3.3 Hadoop Job History Loader

The HadoopJobHistoryLoader in Piggybank loads Hadoop job history files and job xml files from file system. For each MapReduce job, the loader produces a tuple with schema (j:map[], m:map[], r:map[]). The first map in the schema contains job-related entries. Here are some of important key names in the map:

PIG_SCRIPT_ID	USER	SUBMIT_TIME
CLUSTER	HADOOP_VERSION	LAUNCH_TIME
QUEUE_NAME	PIG_VERSION	FINISH_TIME
JOBID	PIG_JOB_FEATURE	TOTAL_MAPS
JOBNAME	PIG_JOB_ALIAS	TOTAL_REDUCES
STATUS	PIG_JOB_PARENTS	

Examples that use the loader to query Pig statistics are shown below.

### 3.4 Examples

Find scripts that generate more than three MapReduce jobs:

```
a = load '/mapred/history/done' using HadoopJobHistoryLoader() as (j:map[], m:map[],
  r:map[]);
b = group a by (j#'PIG_SCRIPT_ID', j#'USER', j#'JOBNAME');
c = foreach b generate group.$1, group.$2, COUNT(a);
d = filter c by $2 > 3;
dump d;
```

Find the running time of each script (in seconds):

```
a = load '/mapred/history/done' using HadoopJobHistoryLoader() as (j:map[], m:map[],
  r:map[]);
b = foreach a generate j#'PIG_SCRIPT_ID' as id, j#'USER' as user, j#'JOBNAME' as
  script_name,
  (Long) j#'SUBMIT_TIME' as start, (Long) j#'FINISH_TIME' as end;
c = group b by (id, user, script_name)
d = foreach c generate group.user, group.script_name, (MAX(b.end) - MIN(b.start))/1000;
```

```
dump d;
```

Find the number of scripts run by user and queue on a cluster:

```
a = load '/mapred/history/done' using HadoopJobHistoryLoader() as (j:map[], m:map[],
  r:map[]);
b = foreach a generate j#'PIG_SCRIPT_ID' as id, j#'USER' as user, j#'QUEUE_NAME' as queue;
c = group b by (id, user, queue) parallel 10;
d = foreach c generate group.user, group.queue, COUNT(b);
dump d;
```

Find scripts that have failed jobs:

```
a = load '/mapred/history/done' using HadoopJobHistoryLoader() as (j:map[], m:map[],
  r:map[]);
b = foreach a generate (Chararray) j#'STATUS' as status, j#'PIG_SCRIPT_ID' as id, j#'USER'
  as user, j#'JOBNAME' as script_name, j#'JOBID' as job;
c = filter b by status != 'SUCCESS';
dump c;
```

Find scripts that use only the default parallelism:

```
a = load '/mapred/history/done' using HadoopJobHistoryLoader() as (j:map[], m:map[],
  r:map[]);
b = foreach a generate j#'PIG_SCRIPT_ID' as id, j#'USER' as user, j#'JOBNAME' as
  script_name, (Long) r#'NUMBER_REDUCES' as reduces;
c = group b by (id, user, script_name) parallel 10;
d = foreach c generate group.user, group.script_name, MAX(b.reduces) as max_reduces;
e = filter d by max_reduces == 1;
dump e;
```

## 4 Pig Progress Notification Listener

Pig provides the ability to register a listener to receive event notifications during the execution of a script. Events include MapReduce plan creation, script launch, script progress, script completion, job submit, job start, job completion and job failure.

To register a listener, set the `pig.notification.listener` parameter to the fully qualified class name of an implementation of [org.apache.pig.tools.pigstats.PigProgressNotificationListener](http://org.apache.pig.tools.pigstats.PigProgressNotificationListener). The class must exist on the classpath of the process submitting the Pig job. If the `pig.notification.listener.arg` parameter is set, the value will be passed to a constructor of the implementing class that takes a single String.

## 5 PigUnit

PigUnit is a simple xUnit framework that enables you to easily test your Pig scripts. With PigUnit you can perform unit testing, regression testing, and rapid prototyping. No cluster set up is required if you run Pig in local mode.

## 5.1 Build PigUnit

To compile PigUnit run the command shown below from the Pig trunk. The compile will create the pigunit.jar file.

```
$pig_trunk ant pigunit-jar
```

## 5.2 Run PigUnit

You can run PigUnit using Pig's local mode or mapreduce mode.

### 5.2.1 Local Mode

PigUnit runs in Pig's local mode by default. Local mode is fast and enables you to use your local file system as the HDFS cluster. Local mode does not require a real cluster but a new local one is created each time.

### 5.2.2 Mapreduce Mode

PigUnit also runs in Pig's mapreduce mode. Mapreduce mode requires you to use a Hadoop cluster and HDFS installation. It is enabled when the Java system property pigunit.exectype.cluster is set to any value: e.g. -Dpigunit.exectype.cluster=true or System.getProperties().setProperty("pigunit.exectype.cluster", "true"). The cluster you select must be specified in the CLASSPATH (similar to the HADOOP\_CONF\_DIR variable).

## 5.3 PigUnit Example

Many PigUnit examples are available in the [PigUnit tests](#).

The example included here computes the top N of the most common queries. The Pig script, top\_queries.pig, is similar to the [Query Phrase Popularity](#) in the Pig tutorial. It expects an input a file of queries and a parameter n (n is 2 in our case in order to do a top 2).

Setting up a test for this script is easy because the argument and the input data are specified by two text arrays. It is the same for the expected output of the script that will be compared to the actual result of the execution of the Pig script.

### 5.3.1 Java Test

```
@Test
public void testTop2Queries() {
    String[] args = {
        "n=2",
    };

    PigTest test = new PigTest("top_queries.pig", args);
```

```

String[] input = {
    "yahoo",
    "yahoo",
    "yahoo",
    "twitter",
    "facebook",
    "facebook",
    "linkedin",
};

String[] output = {
    "(yahoo,3)",
    "(facebook,2)",
};

test.assertOutput("data", input, "queries_limit", output);
}

```

### 5.3.2 top\_queries.pig

```

data =
  LOAD 'input'
  AS (query:CHARARRAY);

queries_group =
  GROUP data
  BY query;

queries_count =
  FOREACH queries_group
  GENERATE
    group AS query,
    COUNT(data) AS total;

queries_ordered =
  ORDER queries_count
  BY total DESC, query;

queries_limit =
  LIMIT queries_ordered $n;

STORE queries_limit INTO 'output';

```

### 5.3.3 Run

The test can be executed by JUnit (or any other Java testing framework). It requires:

1. pig.jar
2. pigunit.jar

The test takes about 25s to run and should pass. In case of error (for example change the parameter n to n=3), the diff of output is displayed:

```
junit.framework.ComparisonFailure: null expected:<...ahoo,3)
(facebook,2)[]> but was:<...ahoo,3)
(facebook,2)[
(linkedin,1)]>
    at junit.framework.Assert.assertEquals(Assert.java:81)
    at junit.framework.Assert.assertEquals(Assert.java:87)
    at org.apache.pig.pigunit.PigTest.assertEquals(PigTest.java:272)
```

## 5.4 Troubleshooting Tips

Common problems you may encounter are discussed below.

### 5.4.1 Classpath in Mapreduce Mode

When using PigUnit in mapreduce mode, be sure to include the \$HADOOP\_CONF\_DIR of the cluster in your CLASSPATH.

MiniCluster generates one in build/classes.

```
org.apache.pig.backend.executionengine.ExecException:
ERROR 4010: Cannot find hadoop configurations in classpath
(neither hadoop-site.xml nor core-site.xml was found in the classpath).
If you plan to use local mode, please put -x local option in command line
```

### 5.4.2 UDF jars Not Found

This error means that you are missing some jars in your test environment.

```
WARN util.JarManager: Couldn't find the jar for
org.apache.pig.piggybank.evaluation.string.LOWER, skip it
```

### 5.4.3 Storing Data

Pig currently drops all STORE and DUMP commands. You can tell PigUnit to keep the commands and execute the script:

```
test = new PigTest(PIG_SCRIPT, args);
test.unoverride("STORE");
test.runScript();
```

### 5.4.4 Cache Archive

For cache archive to work, your test environment needs to have the cache archive options specified by Java properties or in an additional XML configuration in its CLASSPATH.

If you use a local cluster, you need to set the required environment variables before starting it:

```
export LD_LIBRARY_PATH=/home/path/to/lib
```

## 5.5 Future Enhancements

Improvements and other components based on PigUnit that could be built later.

For example, we could build a PigTestCase and PigTestSuite on top of PigTest to:

1. Add the notion of workspaces for each test.
2. Remove the boiler plate code appearing when there is more than one test methods.
3. Add a standalone utility that reads test configurations and generates a test report.