

# Apache Pig



- What is Pig
- Motivation
- How is it being used
- Data Model/Architecture
- Components
- Pig Latin by Example

# Pig



- Pig is a simple-to-understand data flow language used in the analysis of large data sets.
- Pig scripts are automatically converted into MapReduce jobs by the Pig interpreter, so you can analyze the data in a Hadoop cluster even if you aren't familiar with MapReduce.

# What is Pig?

**Apache Pig** is a platform for analyzing large data sets. Pig's language, Pig Latin, is a simple query algebra that lets you express data transformations such as merging data sets, filtering them, and applying functions to records or groups of records.

## Hadoop/Pig Architecture



Simple to understand data flow language for analysts familiar with scripting languages

Fast, iterative language with strong MapReduce compilation engine

Rich, multivalued, nested operations performed on large data sets

# Pig is a Strong and Sophisticated Language for Hadoop

Challenge	Solution
MapReduce requires a Java programmer	Opens the systems to users familiar with scripting languages such as Python, PHP, Ruby
MapReduce can require multiple stages to come to a solution	In one test: <ul style="list-style-type: none"><li>• 10 lines of Pig Latin = 200 lines of Java</li><li>• What took 4 hours in java → 15 minutes in Pig Latin</li></ul>
User has to re-invent common functionality (join, filter, etc.)	Provides common operations like join, group, filter, sort
MapReduce has a long development cycle with rigorous testing states	Pig provides Pig Latin that increases productivity x10 and the programmer can leverage things such as nested types



# Uses of Pig



- Rapid prototyping of algorithms for processing large data sets
- Data processing for web search platforms
- Ad hoc queries across large data sets
- Web log processing

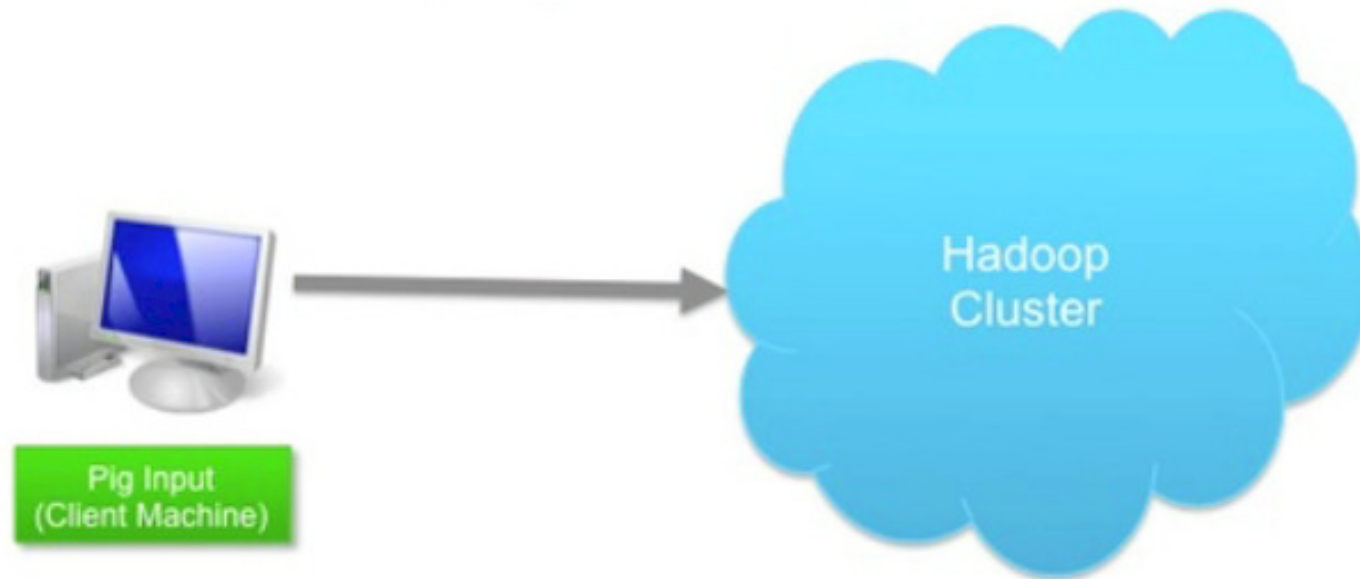
# How to Access Pig



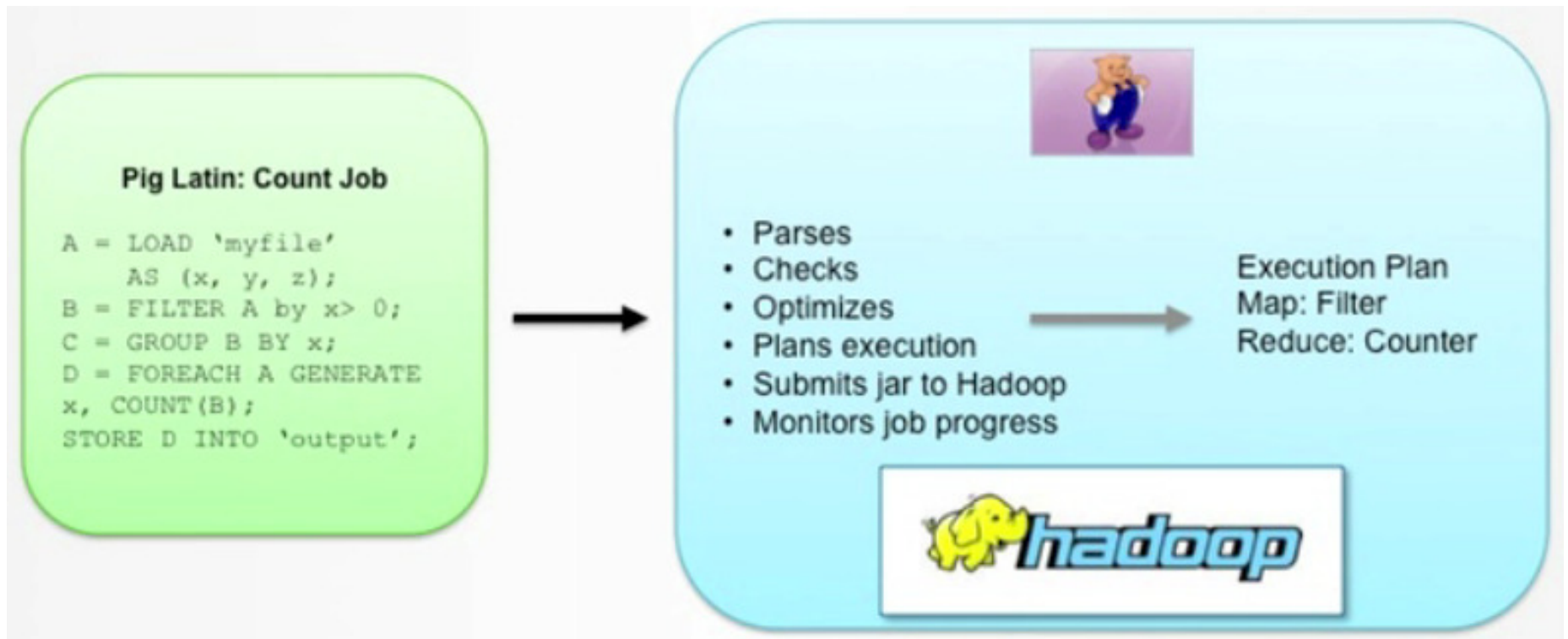
- Grunt, the Pig shell
- Submit a script directly
- PigServer Java class, a JDBC like interface
- PigPen: an eclipse plugin
  - Allows textual & graphical scripting
  - Samples data & shows example data flow

# Pig Resides on User Machine

- Job submitted to cluster & executed on cluster
- No need to install anything extra on cluster



# How does Pig Work?





# Pig Data Types



- Scalar types
  - Int
  - Long
  - Double
  - Chararray
  - Bytearray
- Complex types
  - Map: associative array
  - Tuple: ordered list of data, elements may be of any scalar or complex type
  - Bag: unordered collection of tuples

# How to Load Data



- Welcome to the Pig data loader
  - PigStorage: loads/stores relations using field-delimited text format.
  - BinStorage: loads/stores relations from or to binary files
  - BinaryStorage: loads/stores relations containing only single-field tuples with a value of type bytearray
  - TextLoader: Loads relations from a plain-text format
  - PigDump: Stores relations by writing the toString() representation of tuples, one per line.

# Example: Processing Log Files

- 
- We will be using some HTTP logs from previous lectures.

# We need to start the Pig Shell



- To start the Pig shell (Grunt), start a terminal and run

```
$ cd /usr/share/doc/hadoop-pig/examples/data  
$ pig -x local
```

- Should see a prompt like:

```
grunt>
```

# Aggregation

- Let's count the number of times each user appears in the excite data set

```
log = LOAD 'excited-small.log'  
  AS (user, timestamp, query);  
grp = GROUP log BY user;  
cntd = FOREACH grp GENERATE group, COUNT(log);  
STORE cntd INTO 'output';
```

- Results:

002BB5A52580A8ED	18
005BD9CD3AC6BB38	18

# What is happening in the script?

```
log = LOAD 'excited-small.log'  
  AS (user, timestamp, query);  
grp = GROUP log BY user;  
cntd = FOREACH grp GENERATE group, COUNT(log);  
STORE cntd INTO 'output';
```

- Load
- Group
- ForEach
- Generate
- Count
- Store

# Datasets and Aliases

- Each statement defines a new *dataset*, possibly in terms of existing datasets
- Each dataset is immutable
- Datasets can be given aliases to use later

```
log = LOAD 'excited-small.log' AS (user, timestamp, query);
```

- Note: Use the **dump** command to examine the contents of an alias

```
DUMP log;
```

# Load returns a Tuple



- LOAD statements return a *tuple*. Each tuple has multiple elements, which can be referenced by position or by *name*.

```
log = LOAD 'excited-small.log' AS (user, timestamp, query);
```

- When you perform a DUMP of the contents, there are normally multiple *tuples* which are referred to as *bags*



# Bags and ForEach

- The FOREACH...GENERATE statement iterates over the members of a bag

```
Username = FOREACH log GENERATE user;
```

- The result of a FOREACH is another bag
- Elements are named as in the input bag

# Positional Reference

- The following creates identical output data

```
U usernames = FOREACH log GENERATE $0;
```

- ...But the elements of **usernames** aren't named "**user**" – unless you do this:

```
U usernames = FOREACH log GENERATE $0 as user;
```

# Grouping

- In Pig grouping is a separate operation from applying aggregate functions
- In the earlier example, the output of the group statement is (key, bag), where key is the group key and bag contains a tuple for every record with that key
- For example:

Alan 1

Bob 9 → alan, {(alan, 1), (alan, 3)}

Alan 3 → bob, {(bob, 0)}

# Grouping and Types

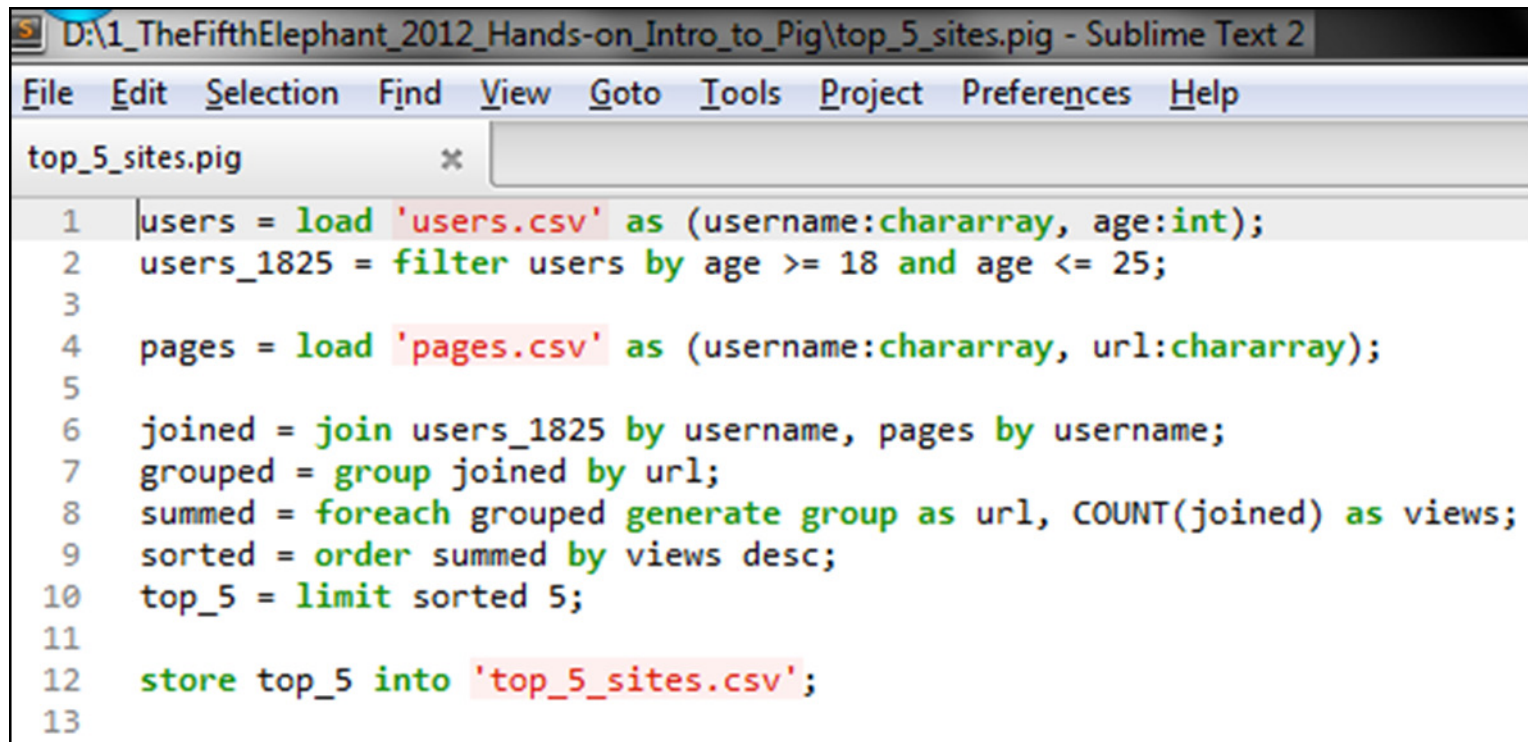
- GROUP BY makes an output bag containing tuples, containing more bags

```
Gprd = GROUP log BY user;
```

- In: BagOf(user, query, time)
- Out: BagOf(**group**, BagOf(user, query, time), named log)
- The grouping item is always named “**group**”

# MapReduce is difficult to program

- One frequent complaint about MapReduce is that it's difficult to program.
- When you first think through a data processing task, you may think about it in terms of data flow operations, such as loops and filters.
- However, as you implement the program in MapReduce, you'll have to think at the level of mapper and reducer functions and job chaining.
- Pig is a Hadoop extension that simplifies Hadoop programming by giving you a high-level data processing language while keeping Hadoop's simple scalability and reliability.



The image shows a screenshot of the Sublime Text 2 editor. The title bar at the top reads "D:\1\_TheFifthElephant\_2012\_Hands-on\_Intro\_to\_Pig\top\_5\_sites.pig - Sublime Text 2". Below the title bar is a menu bar with the following options: File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The editor window displays a file named "top\_5\_sites.pig". The code is a Pig script with 13 lines, each numbered on the left. The script loads two CSV files, filters users by age, joins the data by username, groups by url, generates a count of views, orders by view count in descending order, limits to the top 5, and finally stores the result into a new CSV file.

```
1 users = load 'users.csv' as (username:chararray, age:int);
2 users_1825 = filter users by age >= 18 and age <= 25;
3
4 pages = load 'pages.csv' as (username:chararray, url:chararray);
5
6 joined = join users_1825 by username, pages by username;
7 grouped = group joined by url;
8 summed = foreach grouped generate group as url, COUNT(joined) as views;
9 sorted = order summed by views desc;
10 top_5 = limit sorted 5;
11
12 store top_5 into 'top_5_sites.csv';
13
```

```
// Item ID // Only output the first 100 records while (count < 100 && !user.isAuthenticated()) {
// accordingly. List<String> first = new ArrayList<String>(); user.logout(msg, User.Session()); }
List<String> second = new ArrayList<String>(); count++; }
while (!user.isAuthenticated()) { }
Task t = User.Session(); }
String value = s.toString(); }
if (value.charAt(0) == '1') { public static void main(String[] args) throws IOException {
first.add(value); JobConf jc = new JobConf(MainExample.class);
} }
else second.add(value); JobConf jc = new JobConf("Find top 100 sites for users");
jc.setOutputFormat(TextOutputFormat.class); }
}
}
```

# Users of Pig



- Yahoo , one of the heaviest user of Hadoop (and a backer of both the Hadoop Core and Pig), runs 40 percent of all its Hadoop jobs with Pig.
- Twitter is also another well-known user of Pig.



## Committers of Pig

YAHOO!

Linked in



Inadco



Autodesk



Source: <http://pig.apache.org/whoweare.html>

Who is using Pig?



Source: <http://wiki.apache.org/pig/PoweredBy>

# Pig Use Cases



- ❑ **Processing many Data Sources**
- ❑ **Data Analysis**
- ❑ **Text Processing**
  - **Structured**
  - **Semi-Structured**
- ❑ **ETL**
- ❑ **Machine Learning**
- ❑ **Advantage of Sampling in any use case**

# Pig in real-world

## LinkedIn

**People You May Know**

-  **Arvind Diwakar**, Program Manager at Target ×  
[Connect](#)
-  **Deepak Chaudhary**, Platform Specialist at Sapient ×  
[Connect](#)
-  **Amit Agrawal**, Software engineer at Apple Inc. ×  
[Connect](#)





[See more »](#)

**Who's Viewed Your Profile?**

- 5** Your profile has been viewed by 5 people in the past 15 days.
- 18** You have shown up in search results 18 times in the past 7 days.


## Twitter

**Who to follow** · [Refresh](#) · [View all](#)


-  **Adam Kinney** @kadamk ×  
Followed by Jimmy Lin and others  
[Follow](#)
-  **Justin**  @shitmydadsays ×  
Followed by Russell Journey and o...  
[Follow](#)
-  **Peter Norvig** @norvig ×  
Followed by fogus and others  
[Follow](#)

[Browse categories](#) · [Find friends](#)

Reporting, ETL, targeted emails & recommendations, spam analysis, ML

- 
- Pig has two major components:
    - ▣ A high-level data processing language called Pig Latin .
    - ▣ A compiler that compiles and runs your Pig Latin script in a choice of evaluation mechanisms. The main evaluation mechanism is Hadoop. Pig also supports a local mode for development purposes.
  - Pig simplifies programming because of the ease of expressing your code in Pig Latin.
  - The compiler helps to automatically exploit optimization opportunities in your script.
  - This frees you from having to tune your program manually.
  - As the Pig compiler improves, your Pig Latin program will also get an automatic speed-up.

# Thinking like a Pig

- 
- Pig has a certain philosophy about its design.
  - We expect ease of use, high performance, and massive scalability from any Hadoop subproject.
  - More unique and crucial to understanding Pig are the design choices of its programming language (a data flow language called Pig Latin), the data types it supports, and its treatment of user-defined functions (UDFs ) as first-class citizens.

## ***Data flow language***

You write Pig Latin programs in a sequence of steps where each step is a single high-level data transformation. The transformations support relational-style operations, such as filter, union, group, and join. An example Pig Latin program that processes a search query log may look like

```
log  = LOAD 'excite-small.log' AS (user, time, query);
grpd = GROUP log BY user;
cntd = FOREACH grpd GENERATE group, COUNT(log);
DUMP cntd;
```

Even though the operations are relational in style, Pig Latin remains a data flow language. A data flow language is friendlier to programmers who think in terms of algorithms, which are more naturally expressed by the data and control flows. On the other hand, a declarative language such as SQL is sometimes easier for analysts who prefer to just state the results one expects from a program. Hive is a different Hadoop subproject that targets users who prefer the SQL model.

# Pigs eat anything

- We can summarize Pig's philosophy toward data types in its slogan of "Pigs eat anything." Input data can come in any format.
- Popular formats, such as tab-delimited text files, are natively supported.
- Users can add functions to support other data file formats as well.
- Pig doesn't require metadata or schema on data, but it can take advantage of them if they're provided.
- Pig can operate on data that is relational, nested, semistructured, or unstructured.
- To support this diversity of data, Pig supports complex data types, such as bags and tuples that can be nested to form fairly sophisticated data structures.



# User-defined functions



- Pig was designed with many applications in mind—processing log data, natural language processing, analyzing network graphs, and so forth.
- It's expected that many of the computations will require custom processing.
- Pig is architected from the ground up with support for user-defined functions.
- Knowing how to write UDFs is a big part of learning to use Pig.

# Installing Pig

- As usual, make sure to set JAVA\_HOME to the root of your Java installation, and Windows users should install Cygwin.
- Your Hadoop cluster should already be set up.
- Ideally it's a real cluster in fully distributed mode, although a pseudo-distributed setup is fine for practice.
- You install Pig on your local machine by unpacking the downloaded distribution.
- There's nothing you have to modify on your Hadoop cluster.
- Think of the Pig distribution as a compiler and some development and deployment tools.
- It enhances your MapReduce programming but is otherwise only loosely coupled with the production Hadoop cluster.

# Configuration

- Under the directory where you unpacked Pig, you should create the subdirectories logs and conf (unless they're already there).
- Pig will take custom configuration from files in conf.
- If you are creating the conf directory just now, there's obviously no configuration file, and you'll need to put in conf a new file named pig-env.sh.
- This script is executed when you run Pig, and it can be used to set up environment variables for configuring Pig.
- Besides JAVA\_HOME, the environment variables of particular interest are PIG\_HADOOP\_VERSION and PIG\_CLASSPATH.
- You set these variables to instruct Pig about your Hadoop cluster.
- For example, the following statements in pig-env.sh will tell Pig the version of Hadoop used by the cluster is 0.18, and to add the configuration directory of your local installation of Hadoop to Pig's classpath:
  - ▣ export PIG\_HADOOP\_VERSION=18
  - ▣ export PIG\_CLASSPATH=\$HADOOP\_HOME/conf/

# Hadoop cluster's NameNode and JobTracker.

- We assume HADOOP\_HOME is set to Hadoop's installation directory on your local machine.
- By adding Hadoop's conf directory to Pig's classpath, Pig can automatically pick up the location of your Hadoop cluster's NameNode and JobTracker.
- Instead of using Pig's classpath, you can also specify the location of your Hadoop cluster by creating a pig.properties file.
- This properties file will be under the conf directory you created earlier.
- It should define fs.default.name and mapred.job.tracker, the filesystem (i.e., HDFS's NameNode ) and the location of the JobTracker.
- An example pig.properties file pointing to a Hadoop set:
  - ▣ fs.default.name=hdfs://localhost:9000
  - ▣ mapred.job.tracker=localhost:9001

For the sake of convenience, let's add the Pig installation's `bin` directory to your path. Assume `$PIG_HOME` is pointing to your Pig's installation:

```
export PATH=$PATH:$PIG_HOME/bin
```

With Pig's `bin` directory set as part of your command line path, you can start Pig with the command `pig`. You may want to first see its usage options:

```
pig -help
```

Let's start Pig's interactive shell to see that it's reading the configurations properly.

```
pig
```

```
2009-07-11 22:33:04,797 [main] INFO
```

```
➡ org.apache.pig.backend.hadoop.executionengine.HExecutionEngine -
```

```
➡ Connecting to hadoop file system at: hdfs://localhost:9000
```

```
2009-07-11 22:33:09,533 [main] INFO
```

```
➡ org.apache.pig.backend.hadoop.executionengine.HExecutionEngine -
```

```
➡ Connecting to map-reduce job tracker at: localhost:9001
```

```
grunt>
```

The filesystem and the JobTracker Pig reports should be consistent with your configuration setup. You're now inside Pig's interactive shell, also known as Grunt.

## ***Running Pig***

We can run Pig Latin commands in three ways—via the Grunt interactive shell, through a script file, and as embedded queries inside Java programs. Each way can work in one of two modes—local mode and Hadoop mode. (Hadoop mode is sometimes called Mapreduce mode in the Pig documentation.) At the end of the previous section we’ve entered the Grunt shell running in Hadoop mode.

The Grunt shell allows you to enter Pig commands manually. This is typically used for ad hoc data analysis or during the interactive cycles of program development. Large Pig programs or ones that will be run repeatedly are run in script files. To enter Grunt, use the command `pig`. To run a Pig script, execute the same `pig` command with the file name as the argument, such as `pig myscript.pig`. The convention is to use the `.pig` extension for Pig scripts.

You can think of Pig programs as similar to SQL queries, and Pig provides a `PigServer` class that allows any Java program to execute Pig queries. Conceptually this is analogous to using JDBC to execute SQL queries. Embedded Pig programs is a fairly advanced topic and you can find more details at <http://wiki.apache.org/pig/EmbeddedPig>.



When you run Pig in local mode, you don't use Hadoop at all.<sup>2</sup> Pig commands are compiled to run locally in their own JVM, accessing local files. This is typically used for development purposes, where you can get fast feedback by running locally against a small development data set. Running Pig in Hadoop mode means the compiled Pig program will physically execute in a Hadoop installation. Typically the Hadoop installation is a fully distributed cluster. (The pseudo-distributed Hadoop setup we used in section 10.2 was purely for demonstration. It's rarely used except to debug configurations.) The execution mode is specified to the `pig` command via the `-x` or `-exectype` option. You can enter the Grunt shell in local mode through:

```
pig -x local
```

Entering the Grunt shell in Hadoop mode is

```
pig -x mapreduce
```

or use the `pig` command without arguments, as it chooses the Hadoop mode by default.

## Managing the Grunt shell

In addition to running Pig Latin statements (which we'll look at in a later section), the Grunt shell supports some basic utility commands.<sup>3</sup> Typing `help` will print out a help screen of such utility commands. You exit the Grunt shell with `quit`. You can stop a Hadoop job with the `kill` command followed by the Hadoop job ID. Some Pig parameters are set with the `set` command. For example,

```
grunt> set debug on
grunt> set job.name 'my job'
```

The `debug` parameter states whether debug-level logging is turned on or off. The `job.name` parameter takes a single-quoted string and will use that as the Pig program's Hadoop job name. It's useful to set a meaningful name to easily identify your Pig job in Hadoop's Web UI.

The Grunt shell also supports file utility commands, such as `ls` and `cp`. You can see the full list of utility commands and file commands in table 10.1. The file commands are mostly a subset of the HDFS filesystem shell commands, and their usage should be self-explanatory.

**Table 10.1** Utility and file commands in the Grunt shell

Utility commands	<code>help</code> <code>quit</code> <code>kill <i>jobid</i></code> <code>set debug [on off]</code> <code>set job.name '<i>jobname</i>'</code>
File commands	<code>cat</code> , <code>cd</code> , <code>copyFromLocal</code> , <code>copyToLocal</code> , <code>cp</code> , <code>ls</code> , <code>mkdir</code> , <code>mv</code> , <code>pwd</code> , <code>rm</code> , <code>rmf</code> , <code>exec</code> , <code>run</code>



## ***Learning Pig Latin through Grunt***

Before formally describing Pig's data types and data processing operators, let's run a few commands in the Grunt shell to get a feel for how to process data in Pig. For the purpose of learning, it's more convenient to run Grunt in local mode:

```
pig -x local
```

You may want to first try some of the file commands, such as `pwd` and `ls`, to orient yourself around the filesystem.

Let's look at some data. We'll later reuse the patent data we introduced in chapter 4, but for now let's dig into an interesting data set of query logs from the Excite search engine. This data set already comes with the Pig installation, and it's in the file `tutorial/data/excite-small.log` under the Pig installation directory. The data comes in a three-column, tab-separated format. The first column is an anonymized user ID. The second column is a Unix timestamp, and the third is the search query. A decidedly non-random sample from the 4,500 records of this file looks like

3F8AAC2372F6941C	970916093724	minors in possession
C5460576B58BB1CC	970916194352	hacking telenet
9E1707EE57C96C1E	970916073214	buffalo mob crime family
06878125BE78B42C	970916183900	how to make ecstasy

From within Grunt, enter the following statement to load this data into an "alias" (i.e., variable) called `log`.

```
grunt> log = LOAD 'tutorial/data/excite-small.log' AS (user, time, query);
```

Note that nothing seems to have happened after you entered the statement. In the Grunt shell, Pig parses your statements but doesn't physically execute them until you use a `DUMP` or `STORE` command to ask for the results. The `DUMP` command prints out the content of an alias whereas the `STORE` command stores the content to a file.

# DUMP or STORE

- ❑ The fact that Pig doesn't physically execute any command until you explicitly request some end result will make sense once you remember that we're processing large data sets.
- ❑ There's no memory space to "load" the data, and in any case we want to verify the logic of the execution plan before spending the time and resources to physically execute it.
- ❑ We use the DUMP command usually only for development.
- ❑ Most often you'll STORE significant results into a directory.
- ❑ (Like Hadoop, Pig will automatically partition the data into files named part-nnnnnn.)
- ❑ When you DUMP an alias, you should be sure that its content is small enough to be reasonably printed to screen.
- ❑ The LIMIT command allows you to specify how many tuples (rows) to return back.
- ❑ For example, to see four tuples of log
  - ❑ `grunt> lmt = LIMIT log 4;`
  - ❑ `grunt> DUMP lmt;`
  - ❑ `(2A9EABFB35F5B954,970916105432L,+md foods +proteins)`
  - ❑ `(BED75271605EBD0C,970916001949L,yahoo chat)`
  - ❑ `(BED75271605EBD0C,970916001954L,yahoo chat)`
  - ❑ `(BED75271605EBD0C,970916003523L,yahoo chat)`

**Table 10.2** Data read/write operators in Pig Latin

LOAD	<pre>alias = LOAD 'file' [USING function] [AS schema];</pre> <p>Load data from a file into a relation. Uses the <code>PigStorage</code> load function as default unless specified otherwise with the <code>USING</code> option. The data can be given a schema using the <code>AS</code> option.</p>
LIMIT	<pre>alias = LIMIT alias n;</pre> <p>Limit the number of tuples to <math>n</math>. When used right after <code>alias</code> was processed by an <code>ORDER</code> operator, <code>LIMIT</code> returns the first <math>n</math> tuples. Otherwise there's no guarantee which tuples are returned. The <code>LIMIT</code> operator defies categorization because it's certainly not a read/write operator but it's not a true relational operator either. We include it here for the practical reason that a reader looking up the <code>DUMP</code> operator, explained later, will remember to use the <code>LIMIT</code> operator right before it.</p>
DUMP	<pre>DUMP alias;</pre> <p>Display the content of a relation. Use mainly for debugging. The relation should be small enough for printing on screen. You can apply the <code>LIMIT</code> operation on an alias to make sure it's small enough for display.</p>
STORE	<pre>STORE alias INTO 'directory' [USING function];</pre> <p>Store data from a relation into a directory. The directory must not exist when this command is executed. Pig will create the directory and store the relation in files named <code>part-<i>nnnnn</i></code> in it. Uses the <code>PigStorage</code> store function as default unless specified otherwise with the <code>USING</code> option.</p>