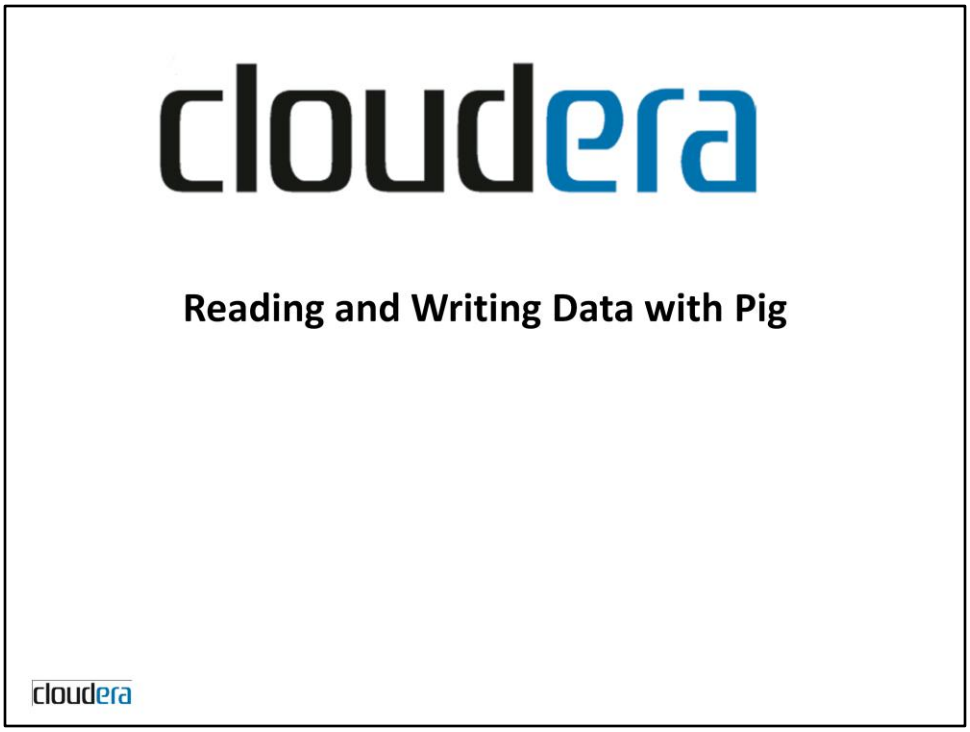
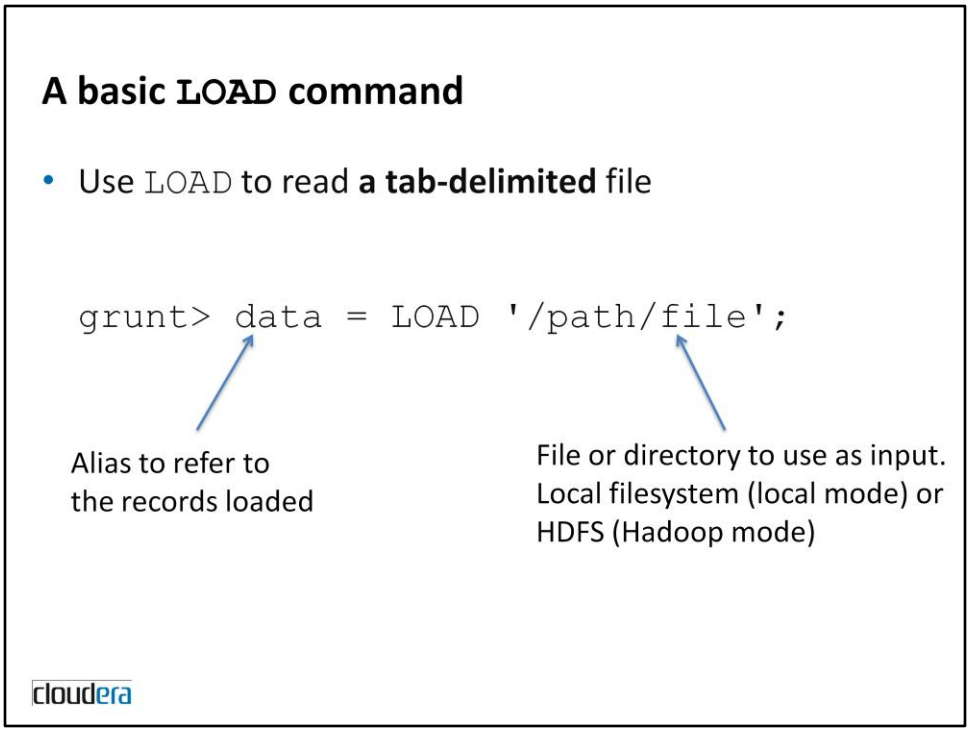


In this chapter, you will learn

- How to read data into a Pig program
- How to add schema information in a Pig program
- How to write data from a Pig program

cloudera



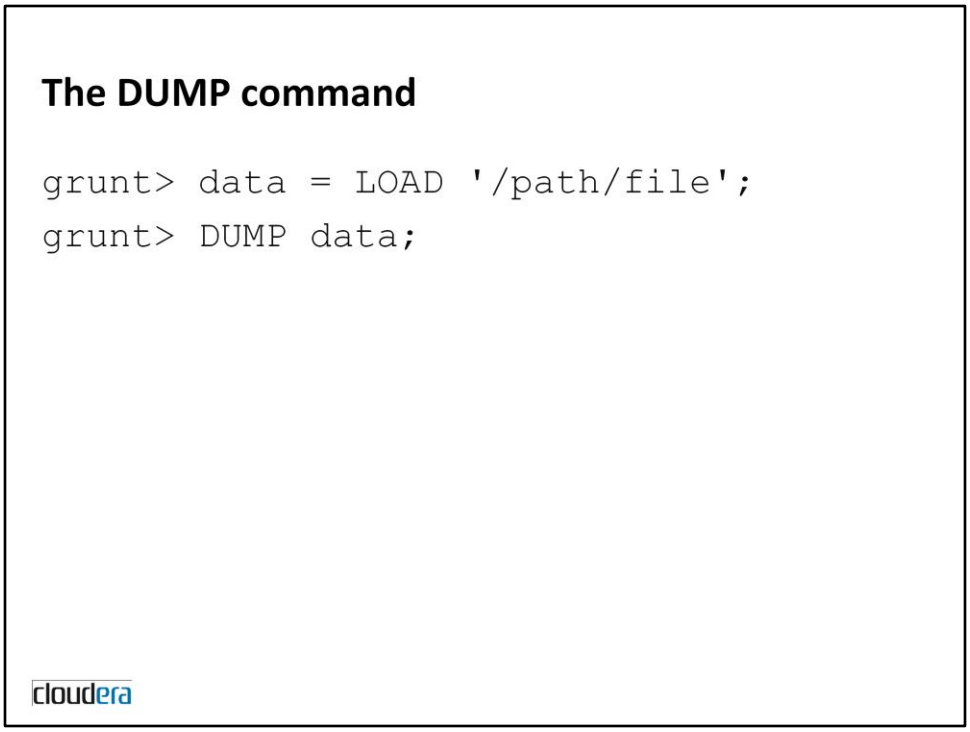
The `LOAD` command takes a filename and optionally information about the data format. The simplest LOAD statement looks like this:

```
grunt> data = LOAD '/path/file';
```

In the above command, `data` is an *alias* representing the records in the file or files. A set of records is also called a *relation*. The path will read from the local filesystem if Pig was invoked in local mode, otherwise the path is treated as an HDFS location. A relative path can also be used.

Although the command is called LOAD, it does not actually "load" data. Instead, it will read the file, when necessary. It does not read the file immediately (in fact, it does not even check if the file exists at this time). Instead, when the Pig program does something that requires the data (for example, store results), the data will then be read.

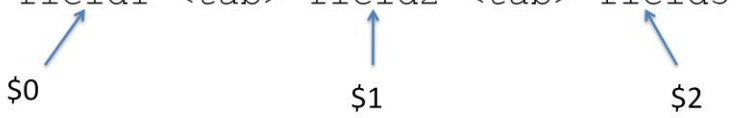
Note: there must be a space after the equal sign. "`data =LOAD`" is illegal syntax.



The `DUMP` command will print a relation to the console. This is very useful for debugging purposes. This works best if the amount of data is small. For larger data sets, we will later discuss storing data in files.

Accessing columns

- By default, LOAD will read the tab-delimited fields into non-typed aliases `$0, $1, $2,` etc
- E.g.,



cloudera

By default, LOAD assumes the data is tab-delimited and it assigns the first field the alias `$0`, the second field `$1`, etc. They are not given a data type. Using such generic field names possibly makes the code difficult to read. Also, it is not obvious to the reader what the data represents. Is `field1` a bunch of characters representing a username or could it be a numeric id number?

Describing the fields

- Define column names:

```
data = LOAD 'file' AS (id, name);
```

- Adding data types:

```
data = LOAD 'file' AS
   (id:int, name:chararray);
```

cloudera

Instead of using the default column names ($0, $1, etc), an AS clause can be added. The list of column names in the AS clause should match the number of columns in the data.

Additionally, columns can be assigned a data type. It is possible to assign data types to none, some or all of the columns. If no data type is given, the fields are read as byte arrays (though they can change later if an implicit or explicit cast is performed).

Pig data types

Type	Example
int	42
long	42L
float	42.0F
double	42.0
chararray	hello
bytearray	
tuple	(123,dcutting)
bag	{(123,dcutting),(124)}
map	[key#value]

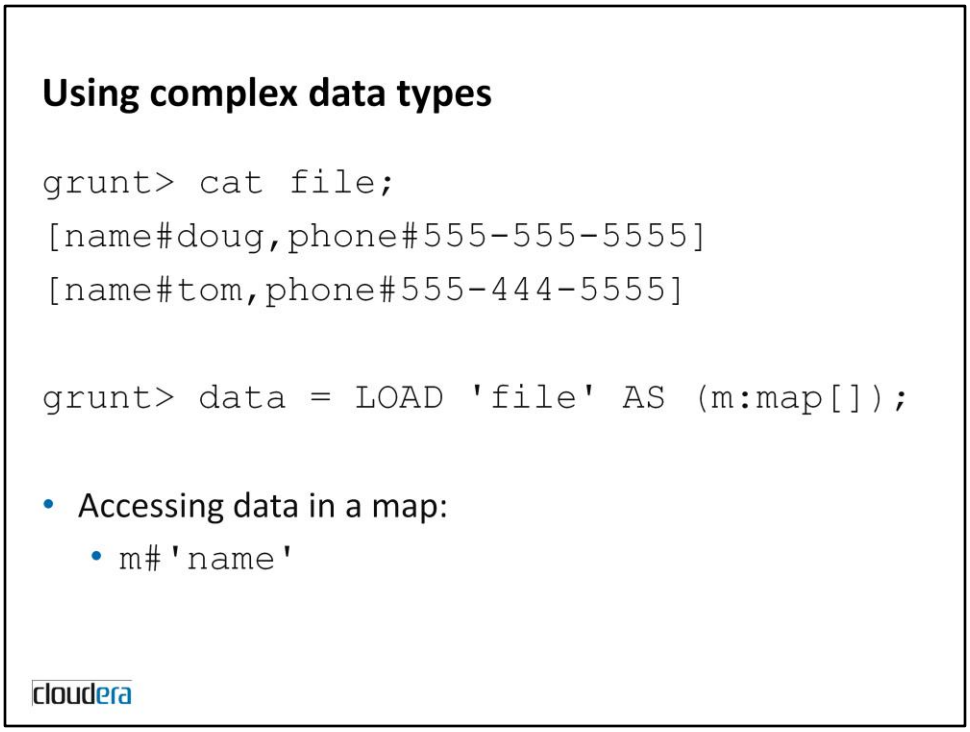
cloudera

There are six simple data types in Pig. They hold scalars (single values).

Simple Types	**Description**	**Example**
int	signed 32-bit integer	42
long	signed 64-bit integer	42L
float	32-bit floating point	42.0F
double	64-bit floating point	42.0
chararray	UTF8 String	hello world
bytearray	byte array (blob)	

There are also three complex data types: tuple, bag and map:

Complex Types	**Description**	**Examples**
tuple	an ordered set of fields	(123) or (123,dcutting)
bag	a collection of tuples	{(123,dcutting),(124)}
map	a set of key/value pairs	[name#doug]



You can load complex data types from a file or files. For example, if a file contained a set of tuples:

```
grunt> cat file;
(doug,42)
(tom,99)
```

This data could be loaded as so:

```
grunt>data = load 'file' AS (record: tuple (f1:chararray,
f2:int));
```

Similarly, a file can contain key/value pairs which may be loaded into a map. Maps are always enclosed in brackets and use # between the key and value:

```
grunt> cat file;
[name#doug,phone#555-555-5555]
[name#tom,phone#555-444-5555]

grunt> data = LOAD 'file' AS (m:map[]);
```

Note, the keyword map is optional. The above statement could also be written as:

```
grunt> data = LOAD 'file' AS (m:[]);
```

DESCRIBE

- Use `DESCRIBE` to view the schema:

```
grunt> DESCRIBE data;
data: {id: int,name: chararray}
```

cloudera

`DESCRIBE` is used to view the schema of a relation. If an AS clause was not specified during the LOAD, then DESCRIBE may return "Schema for records unknown". If column names were given but not data types, then you will get bytearray as the data type:

```
data: {id: bytearray,name: bytearray}
```

Reading various data formats

Loader	Data format	Example
Default	Tab-delimited	`data = LOAD 'file' AS (id, name);`
PigStorage	Any delimiter	`data = LOAD 'file' USING PigStorage(',') AS (id, name);`
TextLoader	Text files	`data = LOAD 'file' USING TextLoader();`

cloudera

It is possible to read any type of data using Pig. The default Loader that we've used so far is PigStorage (with a tab as delimiter). This is the behavior of a `LOAD` that does not have a `USING` clause. These two statements are equivalent:

```
data = LOAD 'file' AS (id, name);
data = LOAD 'file' AS (id, name) USING PigStorage('\t');
```

PigStorage can use any delimiter, such as a comma for CSV files.

Pig also has a built-in TextLoader which parses plain text files line-by-line.

For other data formats, it is possible to write custom Loaders. There are also a set of functions in the PiggyBank found here: http://wiki.apache.org/pig/PiggyBank

One of the useful functions in PiggyBank is MyRegExLoader which can take any regular expression to parse records.

Storing results

- `STORE` is the opposite of `LOAD`
- Examples:

```
grunt> STORE data INTO 'outdir';
grunt> STORE data INTO 'outdir'
    USING PigStorage(',');
```

cloudera

The opposite of `LOAD` is `STORE`. `STORE` writes the records to a directory (in HDFS unless running in Pig's local mode). The default output format is **tab-delimited** fields. Similar to the `LOAD` command, a `USING` clause can be added.

Note: Pig will not overwrite a directory. To remove a directory, use `rm <dirname>`.

In this chapter, you have learned

- How to read data into a Pig program
- How to add schema information in a Pig program
- How to write data from a Pig program

cloudera