coda Documentation

Release 0.1.0

Blake Printy

Oct 05, 2017

Contents

CHAPTER 1

Introduction:

Coda is a file system organizer, designed for data scientists who frequently deal with large amounts of heterogeneous data. In this age where data rules all, being able to efficiently search and label those data is paramount to maintaining productivity. Coda allows you to tag files with arbitrary metadata, so that you can stay organized when managing/analyzing large datasets over time.

As a quick example of how coda might be useful for organizing an arbitrary dataset, see the following example (see the Quickstart section for more in-depth documentation):

```
>>> import coda
>>>
>>> # generate a collection of files from a directory
>>> cl = coda.Collection('/path/to/test/data')
>>>
>>> # show all of the files in the structure
>>> print cl
/path/to/test/data/type1.txt
/path/to/test/data/type1.csv
/path/to/test/data/type2.txt
/path/to/test/data/type2.csv
>>>
>>> # print the number files in that collection
>>> print len(collection)
4
>>>
>>> # set properties about the collection
>>> cl.group = 'test'
>>> cl.cohort = 'My Cohort'
>>>
>>> # add the files in the collection to the database
>>> # for tracking and retrieval later
>>> coda.add(cl)
>>>
>>> # do the same with a training dataset
>>> cl = coda.Collection('/path/to/train/data', metadata={'group': 'train'})
>>> coda.add(cl)
```

```
>>>
>>> # wait ... add one more file in a different location to
>>> # the training set
>>> fi = coda.File('/my/special/training/file.csv')
>>> fi.group = 'train'
>>> coda.add(fi)
>>>
>>> # ... later in time ...
>>>
>>> # query all of our training files
>>> cl = coda.find({'group': 'train'})
>>> print cl
/path/to/train/data/type1.txt
/path/to/train/data/type1.csv
/path/to/train/data/type2.txt
/path/to/train/data/type2.csv
/my/special/training/file.csv
>>>
>>> # filter those by csv files
>>> print cl.filter(lambda x: '.csv' in x.name)
/path/to/train/data/type1.csv
/path/to/train/data/type2.csv
/my/special/training/file.csv
>>>
>>> # tag the special file with new metadata
>>> cl.files[-1].special = True
>>> coda.add(cl.files[-1])
>>>
>>> # query it back (for the example)
>>> fi = coda.find_one({'special': True})
>>> print fi.metadata
{'group': 'train', 'special': True}
```

The topics covered in the Quickstart section give a semi-in-depth overview of the spectrum of capabilities provided by coda. For any other inquiries on how you might be able to use this tool, or if any part of it is left un- or under-documented, contact the developers on GitHub.

CHAPTER 2

Contents:

Installation

Through pip

To install the latest stable release via pip, run:

```
$ pip install coda
```

Via GitHub

To install the bleeding-edge version of the project:

```
$ git clone http://github.com/bprinty/coda.git
$ cd coda
$ python setup.py install
```

Setting up MongoDB

This application uses a MongoDB backend to store all the file annotation information. In order to use this module, you must have MongoDB installed and running on a port that is tied to the current session (a detailed description of the session can be found in the API section). To install MongoDB, visit their website, and follow the instructions.

After installing MongoDB, start the daemon using:

```
$ mongod
```

If this fails to start, it's usually because the data directory used by mongo currently does not exist. Typically, this can be fixed by the following:

```
$ sudo mkdir -p /data/db
$ sudo chown $USER:$USER -R /data/db
```

If it still fails to start, you're on your own (a.k.a. hit up stack overflow) ...

Questions/Feedback

For questions/feedback about any of this, file an issue in the Github Issue Tracker for this project.

Quickstart

Installation

See the Installation section for details on how to install the project. There are several components to getting coda up and running (coda requires a MongoDB instance), so don't just `pip` your way through the installation, please actually read the Installation section.

Files

`coda.File` objects are used throughout `coda` for managing properties about files, and serve as proxies for querying operations that take place. Generally, a new *`coda.File`* object is instantiated and tagged with metadata whenever a file needs to be added for tracking to the coda database. To instantiate a new file, use:

```
>>> fi = coda.File('/path/to/my/file.txt', metadata={
>>>     'group': 'dev',
>>>     'sample': 'A001'
>>> })
>>> print fi.metadata
{'group': 'dev', 'sample': 'A001'}
```

You can also get and set metadata properties directly on the class:

```
>>> fi = coda.File('/path/to/my/file.txt')
>>> fi.group = 'dev'
>>> fi.sample = 'A001'
>>> print fi.metadata
{'group': 'dev', 'sample': 'A001'}
```

If a file already exists in the database, the metadata property will automatically be populated with the content from the database:

```
>>> fi = coda.File('/path/to/my/file.txt', metadata={
>>>     'group': 'dev',
>>>     'sample': 'A001'
>>> })
>>> coda.add(fi)
>>>
>>> fi2 = coda.File('/path/to/my/file.txt')
>>> print fi2.metadata
{'group': 'dev', 'sample': 'A001'}
```

Additionally, you can query the database for a single `coda.File` object matching parameters if you use the `coda.find_one()` query method:

```
>>> fi = coda.find_one({'group': 'dev'})
>>> print fi
'/path/to/my/file.txt'
```

Collections of Files

`coda.Collection` objects are used throughout `coda` for managing properties about collections of files, and serve as proxies for querying operations that take place. Generally, a new `coda.Collection` object is instantiated to group sets of files together, in order to add/update metadata shared by all `coda.File` objects in the collection, or to perform bulk database updates with all associated files. To instantiate a new collection, you can do it several ways:

1. Instantiating the collection object with a list of `coda.File` objects.
2. Adding `coda.File` or `coda.Collection` objects together.
3. Instantiating the collection object with the path to a directory, where all files in that directory are instantiated as `coda.File` objects within that collection. Or, by
4. Querying the database for a collection of files.

Each of these methods can be used in different contexts, depending on the application. Below are a set of examples that detail each of the ways a `coda.Collection` object can be created:

```
>>> # with file objects
>>> one = coda.File('/path/to/file/one.txt', metadata={
>>>     'group': 'dev',
>>>     'sample': 'A001'
>>> })
>>> two = coda.File('/path/to/file/two.txt', metadata={
>>>     'group': 'dev',
>>>     'sample': 'A002'
>>> })
>>> collection = Collection([one, two])
>>>
>>> # adding file objects together
>>> collection = one + two
>>>
>>> # instantiating with a path
>>> collection = coda.Collection('/path/to/file', metadata={'group': 'dev'})
>>>
>>> # once items are in the database, by querying
>>> coda.add(collection)
>>> same_collection = coda.find({'group': 'dev'})
```

Metadata for a collection will only show the metadata shared by all items in a collection. So, using the example above, the `metadata` property on the object would look like:

```
>>> print collection.metadata
{'group': 'dev'}
>>>
>>> # but, you can still access metadata about each of
>>> # the files individually
>>> print collection.files[0].metadata
{'group': 'A001'}
```

Similarly to `coda.File` objects, you can get and set metadata properties for the entire cohort of files directly on the class. Using the `collection` variable above:

```
>>> collection.group = 'test'
>>> print collection.metadata
{'group': 'test'}
```

For files in the collection that already have entries in the database, metadata will automatically be populated with the content from the database. So, using the same example:

```
>>> print collection[0].metadata
{'group': 'test', 'sample': 'A001'}
>>> coda.add(collection)
>>>
>>> cl2 = coda.Collection('/path/to/file')
>>> print cl2
/path/to/file/one.txt
/path/to/file/two.txt
>>>
>>> print cl2.metadata
{'group': 'test'}
>>>
```

Using the same example, you can query for a `coda.Collection` object matching specific metadata criteria, by using the `coda.find()` query method:

```
>>> cl = coda.find({'group': 'test'})
>>> print cl
/path/to/file/one.txt
/path/to/file/two.txt
```

As an addendum to the functionality provided by MongoDB for querying, you can also filter collections returned by queryies using arbitrary functions:

```
>>> cl = coda.find({'group': 'test'})
>>> print cl.filter(lambda x: x.sample == 'A001')
/path/to/file/one.txt
```

Tracking Files

To add files to the `coda` database for tracking, the `coda.add()` method is used. The `coda.add()` takes a `coda.File` with metadata or a `coda.Collection` object, and stores information about that file (i.e. the path and associated metadata) in the database:

```
>>> # instantiate file objects with metadata
>>> one = coda.File('/path/to/file/one.txt', metadata={
>>>     'group': 'dev',
>>>     'sample': 'A001'
>>> })
>>> two = coda.File('/path/to/file/two.txt', metadata={
>>>     'group': 'dev',
>>>     'sample': 'A002'
>>> })
>>> collection = Collection([one, two])
>>>
>>> # add a single file for tracking
>>> coda.add(one)
>>>
>>> # hold up, we want the whole collection added
>>> coda.add(collection)
>>>
>>> # ... later in time ...
>>>
>>> # query all files in the 'dev' group
>>> cl = coda.find({'group': 'dev'})
>>> print cl == collection
True
>>>
>>> # add a new tag for the 'two' file and add it to the database
>>> two.type = 'datatype'
>>> coda.add(two)
```

Untracking Files

To untrack files and delete them from the `coda` database, the `coda.delete()` method is used. The `coda.delete()` takes a `coda.File` with metadata or a `coda.Collection` object, and deletes all instances of associated files in the database. For example:

```
>>> # query for a file object
>>> fi = coda.find_one({'group': 'dev', 'old': True})
>>> coda.delete(fi)
>>>
>>> # delete a tracked file from the filesystem
>>> fi = coda.File('/path/to/file/one.txt')
>>>
>>> # Delete the file -- if it is already in the database,
>>> # it will be removed. Otherwise, nothing happens. It's also
>>> # worth nothing that this method does not delete the actual file.
>>> coda.delete(fi)
>>>
>>> # delete the 'dev' collection from before
>>> cl = coda.find({'group': 'dev'})
>>> coda.delete(cl)
```

Updating Metadata

To update metadata about a `coda.File` or `coda.Collection`, simply re-add the file (using `coda.add()`) with the updated meatadata. For example:

```
>>> # query a file object
>>> fi = coda.find_one({'group': 'dev', 'special': True})
>>> print fi
/path/to/file/three.txt
>>>
>>> # add new metadata on that object and update the database
>>> fi.special = False
>>> fi.key = 'value'
>>> coda.add(fi)
>>>
>>> # show the new metadata -- as shown before, you can just
>>> # instantiate a file object directly, and the metadata will
>>> # flow implicitly from the database
>>> fi = coda.File('/path/to/file/three.txt')
>>> print fi.metadta
{'group': 'dev', 'special': True, 'special': False, 'key': 'value'}
>>>
>>> # you can similarly update a collection -- for the examples
>>> # below, all files have already been added to the database
>>> cl = coda.Collection('/path/to/file')
>>> print cl.metadata
{'group': 'dev'}
>>> cl.key = 'newvalue'
>>> coda.add(cl)
```

Querying

Once files have been added to the database and tagged with metadata, the `coda.find()` and `coda.find_one()`, can be used to query for files matching specific metadata criteria. These two methods take a dictionary of query parameters as an argument and return either a `coda.File` (`coda.find_one()`) or `coda.Collection` object (`coda.find()`) containing the query results. As an example, to query files with a particular metadata tag:

```
>>> cl = coda.find({'group': 'dev'})
>>> print cl
/path/to/dev/file/one.txt
/path/to/dev/file/two.txt
```

Since you can filter collection objects by arbitrary functions, doing more advanced queryies about file contents is easy:

```
>>> # define more advanced filtering function --
>>> # this example just makes sure the number of lines is
>>> # greater than 50
>>> def my_filter(name):
>>>     with open(name, 'r') as fi:
>>>         length = len(fi.readlines())
>>>     return length > 50
>>>
>>> # query and filter the collection
>>> cl = coda.find({'group': 'dev'}).filter(my_filter)
>>> print cl
/path/to/dev/file/two.txt
```

Querying for single files is similarly as easy:

```
>>> fi = coda.find_one({'group': 'dev'})
>>> print fi
/path/to/dev/file/one.txt
```

As alluded to above, `coda` also provides functionality for implicitly doing the querying. If you already have a file object that you want to know metadata about, instead of using `coda.find_one()` with the `path` parameter, you can just instantiate a `coda.File` object and query the metadata directly. The information is pulled implicitly from the database. For example:

```
>>> fi = coda.File('/path/to/dev/file/one.txt')
>>> print fi.metadata
{'group': 'dev', 'sample': 'A001'}
```

You can also use this method of querying for collections:

```
>>> cl = coda.Collection('/path/to/dev/files')
>>> print cl.metadata
{'group': 'dev'}
```

Finally, since `coda` is using MongoDB for storing the metadata, when performing queries with *`coda.find()`* and *`coda.find_one()`*, you can use MongoDB query parameters to do more advanced querying on data:

```
>>> cl = coda.find({'$or': [{'group': 'dev'}, {'group': 'test'}]})
```

Command-Line Use

High-level components of the functionality described above is also accessible via the `coda` command-line entry point. Using the entry point, you can add, delete, and tag files or collections of files. Below are examples of the API:

```
~$ # add a file for tracking to the database
~$ coda add /path/to/file.txt
~$
~$ # add a collection of files for tracking
~$ coda add /path/to/directory/
```

To tag a file or collection with specific metadata, use the `tag` subcommand:

```
~$ # format: coda tag <file> <key> <value>
~$ coda tag /path/to/file.txt extension txt
```

To list all of the tracked files in the current directory, use the `list` subcommand:

```
~$ # format: coda list [<path>]
~$ coda list
/path/to/file.txt
```

To remove a file from tracking, use the `delete` subcommand:

```
~$ coda delete /path/to/file.txt
```

To show all metadata about a file, use the `show` subcommand:

```
~$ coda show /path/to/file.txt
/path/to/file.txt
{
    "extension": "txt"
}
```

To find files matching metadata search criteria, use the `find` command:

```
~$ # format: coda find <key> <value>
~$ coda find extension txt
/path/to/file.txt
```

For more information, check out the command-line help information:

```
~$ coda -h
```

API

Configuration

class `coda.db.`**`Session`**(*host='localhost'*, *port=27017*, *write=True*, *dbname='coda'*)

Object for managing connection to internal database.

Parameters

- **`host`** (*`str`*) – Host with database to connect to.
- **`port`** (*`int`*) – Port to connect to database with.
- **`write`** (*`bool`*) – Whether or not to allow writing to the database.
- **`dbname`** (*`str`*) – Name of database to use.

`db`

Internal property for managing connection to mongodb database.

`options`

Return JSON with options on the session.

`coda.db.`**`options`**(**args*, ***kwargs*)

Set options for the current session.

Parameters **`kwargs`** (*`dict`*) – List of arbitrary config items to set.

Examples

```
>>> # set options to defaults
>>> coda.options()
>>> coda.find_one({'name': 'test'}).path
'/file/on/localhost/server'
>>>
>>> # connect to a database for a different host
>>> coda.options({'host': 'remote'})
>>> coda.find_one({'name': 'test'}).path
'/file/on/remote/server'
```

Files and Collections

class `coda.`**`File`**(*path*, *metadata={}*)

Abstract class for file object.

Parameters

- **`path`** (*`list`*) – List of file object to manage.
- **`metadata`** (*`dict`*) – Dictionary with common metadata for collection, specified a priori.

`__add__`(*other*)

Addition operator for files or collections. Using this, you can add File objects to other File objects to form a Collection, or add File objects to Collection objects to form a new Collection.

Examples

```
>>> # add files to create collection
>>> one = coda.File('/file/one.txt')
>>> two = coda.File('/file/two.txt')
>>> collection = one + two
>>> print collection
'/file/one.txt'
'/file/two.txt'
>>>
>>> # add file to collection to create new collection'
>>> three = coda.File('/file/three.txt')
>>> collection = three + collection
>>> print collection
'/file/three.txt'
'/file/one.txt'
'/file/two.txt'
```

`__contains__`(*item*)
Check if specified string exists in file name.

`__eq__`(*other*)
Test equality for File objects.

`__getattr__`(*name*)
Proxy for accessing metadata directly as a property on the class.

`__getitem__`(*name*)
Proxy for accessing metadata directly as a property on the class.

`__gt__`(*other*)
Comparison operator. Compares left and right file paths alphanumerically.

`__lt__`(*other*)
Comparison operator. Compares left and right file paths alphanumerically.

`__repr__`()
Return string representation for file (file path).

`__setattr__`(*name*, *value*)
Proxy for setting metadata directly as a property on the class.

`__str__`()
Return string representation for file (file path).

`extension`
Return extension for file.

`location`
Return dirname for file.

`metadata`
Proxy for returning metadata – if the file exists in the database, then pull metadata for it if none already exists. If metadata exists for the object, then return that.

`name`
Return basename for file.

class `coda.`**`Collection`**(*files*, *metadata={}*)
Abstract class for collection of file objects.

Parameters

- **files** (*list*) – List of file objects to manage, or path to directory to generate collection from.
- **metadata** (*dict*) – Dictionary with common metadata for collection, specified a priori.

__add__ (*other*)

Addition operator for collections or files. Using this, you can add Collection objects to other Colletion objects to form a Collection, or add Collection objects to File objects to form a new Collection.

Examples

```
>>> # add files to create collection
>>> one = coda.File('/file/one.txt')
>>> two = coda.File('/file/two.txt')
>>> onetwo = one + two
>>> three = coda.File('/file/three.txt')
>>> four = coda.File('/file/four.txt')
>>> threefour = three + four
>>>
>>> # add collection objects to create new collection
>>> print onetwo + threefour
'/file/one.txt'
'/file/two.txt'
'/file/three.txt'
'/file/four.txt'
>>>
>>> # add collection to file object to create new collection
>>> print onetwo + three
'/file/one.txt'
'/file/two.txt'
'/file/three.txt'
```

__contains__ (*item*)

Check if item exists in file set. Input item should be a File object.

__eq__ (*other*)

Compare equality for collections.

__getattr__ (*name*)

Proxy for accessing metadata directly as a property on the class.

__getitem__ (*item*)

Proxy for accessing metadata directly as a property on the class.

__gt__ (*other*)

Compare collection objects by number of files in the collections.

__iter__ ()

Iterator for collection object. Iterates by returning each file.

__len__ ()

Return length of collection object (number of files in collection).

__lt__ (*other*)

Compare collection objects by number of files in the collections.

__repr__ ()

Return string representation for collection (list of file paths).

`__setattr__`(*name*, *value*)

Proxy for setting metadata directly as a property on the class.

`__str__`()

Return string representation for collection (list of file paths).

`__sub__`(*other*)

Subtraction operator for collections or files. Using this, you can subtract Collection objects from other Colletion objects to form a Collection with the difference in files, or subtract File objects from Collection objects to return a new Collection without the File object.

Examples

```
>>> # add files to create collection
>>> one = coda.File('/file/one.txt')
>>> two = coda.File('/file/two.txt')
>>> onetwo = one + two
>>> three = coda.File('/file/three.txt')
>>> onetwothree = onetwo + three
>>>
>>> # subtract collection objects to create new collection
>>> print onetwothree - onetwo
'/file/three.txt'
>>>
>>> # subtract file from collection object to create new collection
>>> print onetwothree - three
'/file/one.txt'
'/file/two.txt'
```

`add_metadata`(**args*, ***kwargs*)

Add metadata for all objects in the collection.

`filelist`

Return list with full paths to files in collection.

`filter`(*func=<function <lambda>>*)

Filter collection using specified function. This function allows for filtering files from collection objects by an arbitrary operator. This could be used for filtering more specifically by existing metadata tags, or by more complex methods that read in the file and perform some operation to it.

Parameters **func** (*function*) – Function to filter with.

Examples

```
>>> # query collection for tag
>>> cl = coda.find({'group': 'testing'})
>>>
>>> # query file by data_type tag (assuming tags exist)
>>> cl.filter(lambda x: x.data_type in ['csv', 'txt'])
```

`metadata`

If no metadata is initially specified for a file, query the database for metadata about the specified file.

Querying

`coda.`**`find`**(*query*)

Search database for files with specified metadata.

Parameters **`query`** (*`dict`*) – Dictionary with query parameters.

Returns Collection object with results.

Return type *Collection*

Examples

```
>>> # assuming the database has already been populated
>>> print coda.find({'type': 'test'})
'/my/testing/file/one.txt'
'/my/testing/file/two.txt'
>>>
>>> # assuming 'count' represents line count in the file
>>> print coda.find({'type': 'test', 'count': {'$lt': 30}})
'/my/testing/file/two.txt'
>>>
>>> # using the filter() method on collections instead
>>> print coda.find({'type': 'test'}).filter(lambda x: x.count < 30)
'/my/testing/file/two.txt'
```

`coda.`**`find_one`**(*query*)

Search database for one file with specified metadata.

Parameters **`query`** (*`dict`*) – Dictionary with query parameters.

Returns File object with results.

Return type *File*

Examples

```
>>> # assuming the database has already been populated
>>> print coda.find_one({'type': 'test'})
'/my/testing/file/one.txt'
>>>
>>> # assuming 'count' represents line count in the file
>>> print coda.find({'type': 'test', 'count': {'$lt': 30}})
'/my/testing/file/two.txt'
```

`coda.`**`add`**(*obj*)

Add file object or collection object to database.

Parameters **`obj`** (`File, Collection`) – File or collection of files to add.

Examples

```
>>> # instantiate File object and add metadata
>>> fi = coda.File('/path/to/test/file.txt')
>>> fi.type = 'test'
```

```
>>>
>>> # add file to database
>>> coda.add(fi)
>>>
>>> # instantiate directory as Collection with common metadata
>>> cl = coda.Collection('/path/to/test/dir/')
>>> cl.type = 'test'
>>> coda.add(cl)
```

`coda.`**`delete`**`(`*obj*`)`

Delete file or collection of files from database.

Parameters **`obj`** (`File`, `Collection`) – File or collection of files to delete.

Examples

```
>>> # instantiate File object and delete
>>> fi = coda.File('/path/to/test/file.txt')
>>> coda.delete(fi)
>>>
>>> # instantiate directory and delete
>>> cl = coda.Collection('/path/to/test/dir/')
>>> coda.delete(cl)
>>>
>>> # query by metadata and delete entries
>>> cl = coda.find({'type': 'testing'})
>>> coda.delete(cl)
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index