nbdime Documentation

Release 0.1.0

Martin Sandve Alnæs

March 21, 2016

Contents

NB! The nbdime project and this documentation and is in a very early stage of development and is not usable for any kind of production work yet.

Contents:

CHAPTER 1

Overview of nbdime project

The nbdime project aims to provide tools for diffing and merging of Jupyter notebooks.

TODO: Write a better introduction and overview.

CHAPTER 2

Installing and testing nbdime

2.1 Dependencies

- Python version 2.7.1, 3.3, 3.4, 3.5
- six
- nbformat

Note the requirement 2.7.1, not 2.7.0, this is because 2.7.1 fixes a bug in difflib in an interface-breaking way.

Dependencies for running tests:

- pytest
- pytest-cov

2.2 Install

Use pip to install. See the pip documentation for options. Some examples:

Install requirements for the current user only:

    pip install –user –upgrade -r requirements.txt

Install nbdime for the current user only:

    pip install –user –upgrade .

Make a local developer install for the current user only:

    pip install –user –upgrade -e .

2.3 Testing

See latest build, test and coverage status at:

    https://travis-ci.org/martinal/nbdime https://coveralls.io/github/martinal/nbdime?branch=master

To run tests, locally, simply run

    py.test

from the project root. If you have python 2 and python 3 installed, you may need to run

```
python3 -m pytest
```

to run the tests with python 3. See the pytest documentation for more options.

If you have notebooks with interesting merge challenges, please consider contributing them to nbdime as test cases!

CHAPTER 3

Commandline interface

Nbdime provides three CLI commands. See

```
nbdiff –help nbpatch –help nbmerge –help
```

for usage details.

CHAPTER 4

Description of the diff representation in nbdime

Note: The diff format herein is still considered experimental until development stabilizes. If you have objections or opinions on the format, please raise them ASAP while the project is in its early stages.

In nbdime, the objects to diff are json-compatible nested structures of dicts (with string keys) and lists of values with heterogeneous types (strings, ints, floats). The difference between these objects will itself be represented as a json-compatible object in a format described below.

4.1 Diff format basics

A diff object represents the difference B-A between two objects A and B as a list of operations (ops) to apply to A to obtain B. Each operation is represented as a dict with at least two items:

{ “op”: <opname>, “key”: <key> }

The objects A and B are either mappings (dicts) or sequences (lists or strings), and a different set of ops are legal for mappings and sequences. Depending on the op, the operation dict usually contains an additional argument, documented below.

4.2 Diff format for mappings

For mappings, the key is always a string. Valid ops are:

- { “op”: “remove”, “key”: <string> } - delete existing value at key
- { “op”: “add”, “key”: <string>, “value”: <value> } - insert new value at key not previously existing
- { “op”: “replace”, “key”: <string>, “value”: <value> } - replace existing value at key with new value
- { “op”: “patch”, “key”: <string>, “diff”: <diffobject> } - patch existing value at key with another diffobject

4.3 Diff format for sequences (list and string)

For sequences the key is always an integer index. This index is relative to object A of length N. Valid ops are:

- { “op”: “removerange”, “key”: <string>, “length”: <n>} - delete the values A[key:key+length]
- { “op”: “addrange”, “key”: <string>, “valuelist”: <values> } - insert new items from valuelist before A[key], at end if key=len(A)

- { “op”: “patch”, “key”: <string>, “diff”: <diffobject> } - patch existing value at key with another diffobject

4.4 Relation to JSONPatch

The above described diff representation has similarities with the JSONPatch standard but is different in a few ways. JSONPatch contains operations “move”, “copy”, “test” not used by nbdime, and nbdime contains operations “addrange”, “removerange”, and “patch” not in JSONPatch. Instead of providing a recursive “patch” op, JSONPatch uses a deep JSON pointer based “path” item in each operation instead of the “key” item nbdime uses. This way JSONPatch can represent the diff object as a single list instead of the ‘tree’ of lists that nbdime uses. To convert a nbdime diff object to the JSONPatch format, use the function

from nbdime.diff_format import to_json_patch jp = to_json_patch(diff_obj)

Note that this function is currently a draft and not covered by tests.

4.5 Examples

For examples of concrete diffs, see e.g. the test suite in test_patch.py.

CHAPTER 5

Representing merge results and conflicts

Nbdime implements a three-way merge of Jupyter notebooks and a large subset of generic json objects. The result of a merge operation with a shared origin object base and modified objects local and remote, is a fully or partially merged object plus diff objects between the partially merged objects and the local and remote objects. These two diff objects represent the merge conflicts that could not be automatically resolved.

TODO: Define output formats for the merge operation.

CHAPTER 6

Notebook specific issues

TODO: Document issues covered and plans here.

CHAPTER 7

Use cases and future development plans

Fundamentally, we envision use cases mainly in the categories of a merge command for version control integration, and diff command for inspecting changes and automated regression testing. At the core of it all is the diff algorithms, which must handle not only text in source cells but also a number of data formats based on mime types in output cells.

7.1 Basic diffing use cases

We assume that basic correct diffing is fairly straightforward to implement, but there are still some issues to discuss.

Other tasks (will make issues of these):

- Pretty-printing of diff for commandline output.
- Plugin framework for mime type specific diffing.
- Diffing of common output types (png, svg, etc.)
- Improve fundamental sequence diff algorithm. Current algorithm is based on a brute force O(N^2) longest common subsequence (LCS) algorithm, this will be rewritten in terms of a faster algorithm such as Myers O(ND) LCS based diff algorithm, optionally using Pythons difflib for some use cases where it.

7.2 Version control use cases

Most commonly, cell source is the primary content, and output can presumably be regenerated. Indee, it is not possible to guarantee that merged sources and merged output is consistent or makes any kind of sense.

Some tasks:

- Merge of output cell content is not planned.
- Is it important to track source lines moving between cells?

7.3 Regression testing use cases

CHAPTER 8

Indices and tables

- genindex
- modindex
- search