JsonGit Documentation

Release 0.0.7

John Krauss

March 25, 2012

CONTENTS

Release v0.0.7. (*Installation*)

Use git as a key-value store to store, track and merge arbitrary data in Python.

```
>>> r = jsongit.init('repo')
>>> r.commit('foo', {})
>>> r.checkout('foo', 'bar')
>>> r.commit('foo', {'roses': 'red'})
>>> r.commit('bar', {'violets': 'blue'})
>>> r.merge('foo', 'bar').message
Auto-merge of be92d3dcb6 and dbde44bada from shared parent 5d55214e4f
>>> r.show('foo')
{u'roses': u'red', u'violets': u'blue'}
>>> for commit in r.log('foo'):
...     print(commit)
'foo'='{u'roses': u'red', u'violets': u'blue'}'@fc9e0f3106
'bar'='{u'violets': u'blue'}'@be92d3dcb6
'bar'='{}'@5bb29ad7dc
'foo'='{u'roses': u'red'}'@dbde44bada
'foo'='{}'@5d55214e4f
```

JsonGit layers above the Python packages pygit2 and json_diff to give you logs, merges, diffs, and persistence for any objects that serialize to JSON. It's licensed BSD.

CHAPTER

ONE

FEATURES

- Works with any object that can be serialized as JSON
- Simple key-value store API
- Portable, persistent repositories
- Automatic merging
- Conflict detection
- Key-specific logs and signatures

CHAPTER

TWO

USAGE

Learn how to install and use JsonGit.

2.1 Installation

Installing JsonGit is easy. It's recommended you install within a virtualenv.

JsonGit has been tested with Python 2.7.1 and 2.6.5.

2.1.1 libgit2

Libgit2 is used to build and modify the git repository. You can find instructions for installing it here.

2.1.2 With Pip

If you want to use JsonGit in your project now, pip is the fastest and easiest way to install.

JsonGit is hosted on PyPI, so you can install it in one line:

```
$ pip install jsongit
```

2.1.3 From Source

If you want to hack on JsonGit, you should grab the source.

You can clone it from the GitHub repository:

```
$ git clone https://github.com/talos/jsongit.git
```

Then, install the requirements:

```
$ cd jsongit
$ pip install -r requirements.txt
```

Finally, install jsongit:

```
$ python setup.py install
```

Testing

If you've installed from source, it's easy to make sure everything works.

```
$ pip install nose
$ nosetests
```

2.2 Getting Started

JsonGit is easy to use and, hopefully, fun. If you haven't yet, head over to the *install* section first. If you need specific information on methods or classes, check out the *api*.

2.2.1 Making a Repo

You'll need to initialize a repo before storing any data in it:

```
>>> repo = jsongit.init('repo')
```

This creates a new directory in the same directory as you are running your Python interpreter, and initializes a bare Git repository in it. Bare git repositories are equivalent to the *.git* folder of a regular git repo.

2.2.2 Storing data

Now that you've got a repo, you can put data in it:

```
>>> repo.add('foo', 'my very special bar')
>>> repo.commit()
```

You can commit any python object that will run through `json.dumps()`. Strings, ints, floats, booleans, dicts, lists, and None, are all OK:

```
>>> repo.add('a', 7)
>>> repo.add('b', 7.77)
>>> repo.add('c', True)
>>> repo.add('d', ['foo', 'bar'])
>>> repo.add('e', {'roses': 'red'})
>>> repo.add('f', None)
>>> repo.commit()
```

Any combination thereof is kosher, too:

```
>>> repo.add('dude', {'job': None, 'age': 42, 'likes': ['bowling', 'rug']})
>>> repo.commit()
```

It's oftentimes convenient to add a value and commit simultaneously:

```
>>> repo.commit('fast', 'and easy, too!')
```

This is akin to *git commit -a <file>*. Until you commit a key, modifications applied to it via `Repository.add()` won't be recorded in history.

2.2.3 Retrieving Data

Pulling the data back out is a matter of retrieving the key's value:

```
>>> foo = repo.show('foo')
u'my very special bar'
```

The returned object preserves the original type:

```
>>> type(repo.show('a'))
<type 'int'>
>>> type(repo.show('b'))
<type 'float'>
>>> type(repo.show('c'))
<type 'bool'>
>>> type(repo.show('d'))
<type 'list'>
>>> type(repo.show('e'))
<type 'dict'>
>>> type(repo.show('f'))
<type 'NoneType'>
```

All data is run back through `json.loads()` on the way out:

```
>>> str(repo.show('dude')['job'])
'None'
>>> repo.show('dude')['likes']
['bowling', 'rug']
```

2.2.4 Commit Data

You can retrieve commit information on a key-by-key basis:

```
>>> repo.commit('foo', 'bar', message="leveraging fu")
>>> commit = repo.head('foo')
>>> commit.message
u'leveraging fu'
>>> commit.author.name
u'Jon Q. User'
>>> commit.time
1332438935L
```

2.2.5 Merging Data

Keys can be merged back together if they split from a single commit. First, checkout an existing key into a new key:

```
>>> repo.commit('spoon', {'material': 'silver'})
>>> repo.checkout('spoon', 'fork')
>>> repo.show('fork')
{u'material': u'silver'}
```

Since *fork* and *spoon* share that initial commit, they can be merged later on. Merging returns a `Merge` with information about what happened:

```
>>> repo.commit('spoon', {'material': 'stainless'})
>>> merge = repo.merge('fork', 'spoon')
>>> merge.message
```

```
u'Auto-merge of d0e0aa8061 and ce29b985cf from shared parent d21cb53771'
>>> repo.show('fork')
{u'material': u'stainless'}
```

Intervening changes to *spoon* were applied to *fork*.

2.2.6 Logs

All the modifications to a key are available in its log:

```
>>> repo.commit('president', 'washington')
>>> repo.commit('president', 'adams')
>>> repo.commit('president', 'madison')
>>> log = repo.log('president')
>>> for commit in log:
...     print(commit.data)
...
madison
adams
washington
```

The `Repository.log()` method returns a generator that yields successively deeper commits.

2.2.7 History

By default, `Repository.show()` returns the data from the most recent commit. You can choose to get something from further back on demand:

```
>>> repo.show('president', back=2)
u'washington'
```

Going too far back in time will raise a friendly reminder:

```
>>> repo.show('president', back=300)
IndexError: president has fewer than 300 commits
```

2.2.8 Index

Until you actually commit a key, its value is kept in the index:

```
>>> repo.add('added', 'but not committed!')
>>> repo.index('added')
u'but not committed!'
```

Since there hasn't been a commit, there's nothing to show:

```
>>> repo.show('added')
KeyError: 'There is no key at added'
```

Modifications independent of commits won't appear in your log, either:

```
>>> repo.add('release', 'pet sounds')
>>> repo.commit('release')
>>> repo.add('release', 'smile')
>>> repo.add('release', 'smiley smile')
>>> repo.commit('release')
```

```
>>> for commit in repo.log('release'):
...     print(commit.data)
...
smiley smile
pet sounds
```

CHAPTER

THREE

API DOCUMENTATION

The technical nitty-gritty.

3.1 API

This documentation covers JsonGit's interfaces.

3.1.1 Repository

JsonGit should be used through the public methods of `Repository`. You should use `init()` to obtain the object, not the constructor.

`jsongit.`**`init`**(*path=None*, *repo=None*, ***kwargs*)

Obtain a `Repository`. Either a path to a repo or an existing *pygit2.Repository* must be provided. If the path exists, it will be interpreted as the path to an existing repository; if it does not, a new bare repository will be created there.

```
>>> repo = jsongit.init('path/to/repo')
```

Or, if you want to take advantage of special *pygit2* options:

```
>>> import pygit2
>>> pygit_repo = pygit2.init_repository('path/to/repo', False)
>>> jsongit_repo = jsongit.init(repo=pygit_repo)
```

Parameters

- **path** (*string*) – The path to a repository. If it is a path that does not exist, a new bare git repository will be initialized there. If it is a path that does exist, then the directory will be used as a repository.
- **repo** (`pygit2.Repository`) – An existing repository object.
- **dumps** (*func*) – (optional) An alternate function to use when dumping data. Defaults to `json.dumps()`.
- **loads** (*func*) – (optional) An alternate function to use when loading data. Defaults to `json.loads()`.

Returns A repository reference

Return type `Repository`

class jsongit.models.**Repository**(*repo*, *dumps*, *loads*)

add(*key*, *value*)

Add a value for a key to the working tree, staging it for commit.

```
>>> repo.add('added', 'but not committed!')
>>> repo.index('added')
u'but not committed!'
>>> repo.show('added')
KeyError: 'There is no key at added'
```

Parameters

- **key** (*string*) – The key to add
- **value** (anything that runs through `json.dumps()`) – The value to insert

Raises `NotJsonError` `InvalidKeyError`

checkout(*source*, *dest*, ***kwargs*)

Replace the HEAD reference for dest with a commit that points back to the value at source.

```
>>> repo.commit('spoon', {'material': 'silver'})
>>> repo.checkout('spoon', 'fork')
>>> repo.show('fork')
{u'material': u'silver'}
```

Parameters

- **source** (*string*) – The source key.
- **dest** (*string*) – The destination key
- **author** (*pygit2.Signature*) – (optional) The author of the commit. Defaults to global author.
- **committer** (*pygit2.Signature*) – (optional) The committer of the commit. Will default to global author.

Raises `StagedDataError`

commit(*key=None*, *value=None*, *add=True*, ***kwargs*)

Commit the index to the working tree.

```
>>> repo.add('foo', 'my very special bar')
>>> repo.commit()
>>> foo = repo.show('foo')
u'my very special bar'
```

If a key and value are specified, will add them immediately before committing them.

```
>>> repo.commit('fast', 'and easy, too!')
>>> foo = repo.show('fast')
u'and easy, too!'
```

Parameters

- **key** (*string*) – The key

- **value** (anything that runs through `json.dumps()`) – The value of the key.
- **author** (*pygit2.Signature*) – (optional) The signature for the author of the first commit. Defaults to global author.
- **message** (*string*) – (optional) Message for first commit. Defaults to "adding [key]" if there was no prior value.
- **author** – (optional) The signature for the committer of the first commit. Defaults to git's *–global author.name* and *author.email*.
- **committer** (*pygit2.Signature*) – (optional) The signature for the committer of the first commit. Defaults to author.
- **parents** (list of `Commit`) – (optional) The parents of this commit. Defaults to the last commit for this key if it already exists, or an empty list if not.

Raises `NotJsonError` `InvalidKeyError`

committed(*key*)

Determine whether there is a commit for a key.

```
>>> repo.committed('foo')
False
>>> repo.commit('foo', 'bar')
>>> repo.committed('foo')
True
```

Parameters **key** (*string*) – the key to check

Returns whether there is a commit for the key.

Return type boolean

destroy()

Erase this Git repository entirely. This will remove its directory. Methods called on a repository or its objects after it is destroyed will throw exceptions.

```
>>> repo.destroy()
>>> repo.commit('foo', 'bar')
AttributeError: 'NoneType' object has no attribute 'write'
```

head(*key*, *back=0*)

Get the head commit for a key.

```
>>> repo.commit('foo', 'bar', message="leveraging fu")
>>> commit = repo.head('foo')
>>> commit.message
u'leveraging fu'
>>> commit.author.name
u'Jon Q. User'
>>> commit.time
1332438935L
```

Parameters

- **key** (*string*) – The key to look up.
- **back** (*integer*) – (optional) How many steps back from head to get the commit. Defaults to 0 (the current head).

Returns the data

Return type int, float, NoneType, unicode, boolean, list, or dict

Raises KeyError if there is no entry for key, IndexError if too many steps back are specified.

index(*key*)

Pull the current data for key from the index.

```
>>> repo.add('added', 'but not committed!')
>>> repo.index('added')
u'but not committed!'
```

Parameters **key** (*string*) – the key to get data for

Returns a value

Return type None, unicode, float, int, dict, list, or boolean

log(*key=None*, *commit=None*, *order=<class 'GIT_SORT_TOPOLOGICAL'>*)

Traverse commits from the specified key or commit. Must specify one or the other.

```
>>> repo.commit('president', 'washington')
>>> repo.commit('president', 'adams')
>>> repo.commit('president', 'madison')
>>> log = repo.log('president')
>>> for commit in log:
...     print(commit.data)
...
madison
adams
washington
```

Parameters

- **key** (*string*) – (optional) The key to look up a log for. Will look from the head commit.
- **commit** (`Commit`) – (optional) An explicit commit to look up log for.
- **order** (*number*) – (optional) Flags to order traversal. Valid flags are in `constants`. Defaults to `GIT_SORT_TOPOLOGICAL`

Returns A generator to traverse commits, yielding :class:'Commit <jsongit.wrappers.Commit>'s.

Return type generator

merge(*dest*, *key=None*, *commit=None*, ***kwargs*)

Try to merge two commits together.

```
>>> repo.commit('spoon', {'material': 'silver'})
>>> repo.checkout('spoon', 'fork')
>>> repo.show('fork')
{u'material': u'silver'}
>>> repo.commit('spoon', {'material': 'stainless'})
>>> merge = repo.merge('fork', 'spoon')
>>> merge.message
u'Auto-merge of d0e0aa8061 and ce29b985cf from shared parent d21cb53771'
>>> repo.show('fork')
{u'material': u'stainless'}
```

Parameters

- **dest** (*string*) – the key to receive the merge
- **key** (*string*) – (optional) the key of the merge source, which will use the head commit.
- **commit** (`Commit`) – (optional) the explicit commit to merge
- **author** (*pygit2.Signature*) – (optional) The author of this commit, if one is necessary. Defaults to global author.
- **committer** (*pygit2.Signature*) – (optional) The committer of this commit, if one is necessary. Will default to global author.

Returns The results of the merge operation

Return type `Merge`

remove (*key*, *force=False*)

Remove the head reference to this key, so that it is no longer visible in the repo. Prior commits and blobs remain in the repo, but detached.

```
>>> repo.commit('foo', 'bar')
>>> repo.remove('foo')
>>> repo.committed('foo')
False
>>> repo.staged('foo')
False
```

Parameters

- **key** (*string*) – The key to remove
- **force** (*boolean*) – (optional) Whether to remove the HEAD reference even if there is data staged in the index but not yet committed. If force is true, the index entry will be removed as well.

Raises `StagedDataError jsongit.StagedDataError`

reset (*key*)

Reset the value in the index to its HEAD value.

```
>>> repo.commit('creation', 'eons')
>>> repo.add('creation', 'seven days')
>>> repo.reset('creation')
>>> repo.index('creation')
u'eons'
```

Parameters **key** (*string*) – the key to reset

show (*key*, *back=0*)

Obtain the data at HEAD, or a certain number of steps back, for key.

```
>>> repo.commit('president', 'washington')
>>> repo.commit('president', 'adams')
>>> repo.commit('president', 'madison')
>>> repo.show('president')
u'madison'
>>> repo.show('president', back=2)
u'washington'
```

Parameters

- **key** (*string*) – The key to look up.
- **back** (*integer*) – (optional) How many steps back from head to get the commit. Defaults to 0 (the current head).

Returns the data

Return type int, float, NoneType, unicode, boolean, list, or dict

Raises KeyError if there is no entry for key, IndexError if too many steps back are specified.

staged(*key*)

Determine whether the value in the index differs from the committed value, if there is an entry in the index.

```
>>> repo.staged('huey')
False
>>> repo.add('huey', 'long')
>>> repo.staged('huey')
True
>>> repo.commit()
>>> repo.staged('huey')
False
```

Parameters **key** (*string*) – The key to check

Returns whether the entries are different.

Return type boolean

3.1.2 Commit

class jsongit.wrappers.**Commit**(*repo*, *key*, *data*, *pygit2_commit*)

A wrapper around `pygit2.Commit` linking to a single key in the repo.

author

Returns The author of this commit.

Return type `pygit2.Signature`

committer

Returns The committer of this commit.

Return type `pygit2.Signature`

data

Returns the data associated with this commit.

Return type Boolean, Number, None, String, Dict, or List

hex

Returns The unique 40-character hex representation of this commit's ID.

Return type string

key

Returns the key associated with this commit.

Return type string

message

Returns The message associated with this commit.

Return type string

oid

Returns The unique 20-byte ID of this Commit.

Return type string

repo

Returns The repository of this commit.

Return type Repository

time

Returns The time of this commit.

Return type long

3.1.3 Diffs & Merges

The repository provides an interface for merging. These classes provide methods and properties to investigate merges.

class jsongit.wrappers.**Merge**(*success*, *original*, *merged*, *message*, *result=None*, *conflict=None*)
A class wrapper for the results of a merge operation.

conflict
The Conflict, if the merge was not a success.

merged
The object that was merged in.

message
The message associated with this merge.

original
The original object.

result

Returns the object resulting from this merge, or None if there was a conflict.

success
Whether the merge was a success.

class jsongit.wrappers.**Diff**(*obj1*, *obj2*)
A class to encapsulate differences between two JSON git objects.

append
A dict of appended keys and their values.

apply(*original*)
Return an object modified with the changes in this diff.

Parameters **original** (*list, dict, number, or string*) – the object to apply the diff to.

Returns the modified object

Return type list, dict, number, or string

classmethod **is_json_diff**(*obj*)
Determine whether a dict was produced by JSON diff.

remove
A dict of removed keys and their values.

replace
The diff is simply to replace wholesale.

update
A DiffWrapper

class jsongit.wrappers.**Conflict**(*diff1*, *diff2*)
A class wrapper for the conflict between two diffs.

append
A dict of key append conflict tuples.

remove
A dict of key removal conflict tuples.

replace
A tuple of the two diffs.

update
A dict of key update conflict tuples.

3.1.4 Exceptions

exception jsongit.**NoGlobalSettingError**(*name*)
Raised when the requested global setting does not exist. Subclasses RuntimeError.

exception jsongit.**DifferentRepoError**
This is raised if a merge is attempted on a Commit in a different repo. Subclasses ValueError.

exception jsongit.**InvalidKeyError**
Raised when a non-string or invalid string is used as a key in a repository. Subclasses TypeError.

exception jsongit.**NotJsonError**
Raised when an object that cannot be run through json.dumps() is committed. Subclasses ValueError.

exception jsongit.**StagedDataError**
Raised when an attempt is made to remove a key that has data staged in the index. Subclasses RuntimeError

3.1.5 Utilities

These are convenience methods used internally by JsonGit. They provide some useful abstractions for pygit2.

jsongit.utils.**signature**(*name*, *email*, *time=None*, *offset=None*)
Convenience method to generate pygit2 signatures.

Parameters

- **name** (*string*) – The name in the signature.
- **email** (*string*) – The email in the signature.
- **time** (*int*) – (optional) the time for the signature, in UTC seconds. Defaults to current time.
- **offset** (*int*) – (optional) the time offset for the signature, in minutes. Defaults to the system offset.

Returns a signature

Return type pygit2.Signature

jsongit.utils.**global_config**(*name*)
Find the value of a *git –global* setting.

```
>>> jsongit.global_config('user.name')
'Jon Q. User'
>>> jsongit.global_config('user.email')
'jon.q@user.com'
```

Parameters **name** (*string*) – the name of the setting

Returns the value of the setting

Return type string

Raises NoGlobalSettingError

PYTHON MODULE INDEX

j