

---

# **wardrobe Documentation**

***Release 0.1***

**Benoit Bryon**

April 23, 2016



<b>1</b>	<b>Contents</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	API . . . . .	3
1.3	About wardrobe . . . . .	11
1.4	Contributing to the project . . . . .	12
<b>2</b>	<b>Ressources</b>	<b>15</b>
<b>3</b>	<b>Indices and tables</b>	<b>17</b>
	<b>Python Module Index</b>	<b>19</b>



wardrobe is a Python project that provides a stack-based datastructure: `StackedDict`.

`StackedDict` is a dictionary-like object with additional methods to save the current state (`commit`) and restore it (`reset`).

```
>>> from wardrobe import StackedDict
>>> clark = StackedDict(top='blue bodysuit', bottom='red underpants',
...                     sex_appeal=True)
>>> clark['bottom']
'red underpants'
>>> clark['friend'] = 'Lois'
>>> dict(clark) == {'top': 'blue bodysuit',
...                'bottom': 'red underpants',
...                'friend': 'Lois',
...                'sex_appeal': True}
True
>>> clark.commit()
<wardrobe.stackeddict.StackedDict object at 0x...>
>>> clark.update({'top': 'shirt', 'bottom': 'jeans', 'head': 'glasses'})
>>> del clark['sex_appeal']
>>> dict(clark) == {'top': 'shirt',
...                'bottom': 'jeans',
...                'head': 'glasses',
...                'friend': 'Lois'}
True
>>> clark.reset()
<wardrobe.stackeddict.StackedDict object at 0x...>
>>> dict(clark) == {'top': 'blue bodysuit',
...                'bottom': 'red underpants',
...                'friend': 'Lois',
...                'sex_appeal': True}
True
```



---

## Contents

---

### 1.1 Installation

This code is open-source. See [License](#) for details.

If you want to contribute to the code, you should go to [Contributing to the project](#) documentation.

Install the package with your favorite Python installer. As an example, with pip:

```
pip install PROJECT
```

Then, you should be able to use it!

```
>>> from wardrobe import StackedDict
>>> something = StackedDict()
```

See [API](#) for a detailed usage documentation.

### 1.2 API

This section details wardrobe API. It is automatically generated from sourcecode's documentation.

<a href="#"><i>wardrobe</i></a>	wardrobe package.
<a href="#"><i>wardrobe.stackeddict</i></a>	StackedDict implementation.
<a href="#"><i>wardrobe.exceptions</i></a>	

#### 1.2.1 wardrobe

wardrobe package.

StackedDict class is available at package level, for convenience:

```
>>> from wardrobe import StackedDict
>>> s = StackedDict(a=1, b=2, c=3)
```

Or:

```
>>> from wardrobe import *
>>> s = StackedDict(a=1, b=2, c=3)
```

See [\*wardrobe.stackeddict.StackedDict\*](#) for details.

`wardrobe.package_dir = '/home/docs/checkouts/readthedocs.org/user_builds/wardrobe/envs/stable/local/lib/python2.7/site'`  
Implement [PEP 396](#)

## 1.2.2 wardrobe.stackeddikt

StackedDict implementation.

**exception** `wardrobe.stackeddikt.NoRevisionException`

Exception raised when `reset()` has been called more times than `commit()`.

**args**

**message**

**class** `wardrobe.stackeddikt.StackedDict` (*initial=None, \*\*kwargs*)

Dictionary-like object made of stacked layers.

Instances act like dictionaries.

Calls to `push()` or `pop()` affect (respectively create or delete) one entire layer of the stack.

```
>>> from wardrobe import StackedDict
>>> clark = StackedDict(top='blue bodysuit', bottom='red underpants',
...                     sex_appeal=True)
>>> clark['bottom']
'red underpants'
>>> clark['friend'] = 'Lois'
>>> dict(clark) == {'top': 'blue bodysuit',
...                'bottom': 'red underpants',
...                'friend': 'Lois',
...                'sex_appeal': True}
True
>>> clark.commit()
<wardrobe.stackeddikt.StackedDict object at 0x...>
>>> clark.update({'top': 'shirt', 'bottom': 'jeans', 'head': 'glasses'})
>>> del clark['sex_appeal']
>>> dict(clark) == {'top': 'shirt',
...                'bottom': 'jeans',
...                'head': 'glasses',
...                'friend': 'Lois'}
True
>>> clark.reset()
<wardrobe.stackeddikt.StackedDict object at 0x...>
>>> dict(clark) == {'top': 'blue bodysuit',
...                'bottom': 'red underpants',
...                'friend': 'Lois',
...                'sex_appeal': True}
True
```

**clear()**

Remove all items from the dictionary.

```
>>> s = StackedDict(a=1, b=2, c=3)
>>> s.clear()
>>> dict(s)
{}
```

Affects only current layer.



```
>>> s = StackedDict(a=1, b=2, c=3)
>>> s.commit().update(c='C', d=4, e=5)
>>> s.clear()
>>> dict(s)
{}
>>> silent = s.reset()
>>> dict(s) == dict(a=1, b=2, c=3)
True
```

**commit()**

Save current dictionary state, record next changes in some diff history.

Returns `StackedDict` instance, so that you can chain operations.

Use `reset()` to restore the saved state.

```
>>> s = StackedDict(a=1)
>>> s.commit().update(a='A')
>>> dict(s)
{'a': 'A'}
>>> silent = s.reset()
>>> dict(s)
{'a': 1}
```

**copy()**

Return a shallow copy of instance.

```
>>> s1 = StackedDict(a=1, b=2, c=3)
>>> s2 = s1.copy()
>>> s1 == s2
True
>>> s1 is s2
False
```

**classmethod fromkeys(seq, value=None)**

Create a new `StackedDict` with keys from `seq` and values set to `value`.

`fromkeys()` is a class method that returns a new `StackedDict`. `value` defaults to `None`.

```
>>> s = StackedDict.fromkeys(['a', 'b', 'c'])
>>> filter(lambda x: x is not None, s.values())
[]
```

```
>>> s = StackedDict.fromkeys(['a', 'b', 'c'], 42)
>>> filter(lambda x: x is not 42, s.values())
[]
```

```
>>> s = StackedDict.fromkeys(range(1, 5), 'Hello world!')
>>> filter(lambda x: x != 'Hello world!', s.values())
[]
```

**get(key, default=None)**

Return the value for `key` if `key` is in the dictionary, else `default`.

If `default` is not given, it defaults to `None`, so that this method never raises a `KeyError`.

```
>>> s = StackedDict(a=1)
>>> s.get('a')
1
>>> s.get('b') is None
True
```

```
>>> s.get('b', 2)
2
```

**has\_key** (*key*)

Return True if key is in instance, False otherwise.

Affects global instance, not “only the current layer”.

```
>>> s = StackedDict(a=1, b=2, c=3)
>>> s.has_key('a')
True
>>> s.has_key('b')
True
>>> s.has_key(1)
False
>>> s.commit()
<wardrobe.stackeddict.StackedDict object at 0x...>
>>> s.has_key('a')
True
>>> del s['a']
>>> s.has_key('a')
False
>>> s.commit()
<wardrobe.stackeddict.StackedDict object at 0x...>
>>> s.has_key('a')
False
```

**items** ()

Return a copy of the StackedDict’s list of (key, value) pairs.

```
>>> s = StackedDict(a=1, b=2)
>>> i = s.items()
>>> i.sort()
>>> i
[('a', 1), ('b', 2)]
>>> s.commit().update(c=3)
>>> i = s.items()
>>> i.sort()
>>> i
[('a', 1), ('b', 2), ('c', 3)]
```

**iteritems** ()

Return an iterator over the StackedDict’s (key, value) pairs.

```
>>> s = StackedDict(a=1, b=2)
>>> i = s.iteritems()
>>> i
<generator object iteritems at 0x...>
>>> i = list(i)
>>> i.sort()
>>> i
[('a', 1), ('b', 2)]
>>> s.commit().update(c=3)
>>> i = s.iteritems()
>>> i
<generator object iteritems at 0x...>
>>> i = list(i)
>>> i.sort()
>>> i
[('a', 1), ('b', 2), ('c', 3)]
```

**iterkeys()**

Return an iterator over the StackedDict's keys.

```
>>> s = StackedDict(a=1, b=2, c=3)
>>> i = s.iterkeys()
>>> i
<generator object iterkeys at 0x...>
>>> l = list(i)
>>> l.sort()
>>> l
['a', 'b', 'c']
```

**itervalues()**

Return an iterator over the StackedDict's values.

```
>>> s = StackedDict(a=1, b=2, c=3)
>>> i = s.itervalues()
>>> i
<generator object itervalues at 0x...>
>>> l = list(i)
>>> l.sort()
>>> l
[1, 2, 3]
```

**keys()**

Return iterable on keys.

```
>>> s = StackedDict(a=1, b=2, c=3)
>>> keys = s.keys()
>>> len(keys) == 3
True
>>> 'a' in keys and 'b' in keys and 'c' in keys
True
```

Deleted keys aren't returned... until the layer where the key was deleted is dropped.

```
>>> s = StackedDict(a=1)
>>> s.keys()
['a']
>>> silent = s.commit()
>>> del s['a']
>>> s.keys()
[]
>>> silent = s.commit()
>>> s.keys()
[]
>>> silent = s.reset()
>>> s.keys()
[]
>>> silent = s.reset()
>>> s.keys()
['a']
```

**pop(key, \*args)**

If key is in the dictionary, remove it and return its value, else return default.

```
>>> s = StackedDict(a=1, b=2, c=3)
>>> s.pop('a')
```

```
1
>>> 'a' in s
False
>>> s.pop('a', 'A')
'A'
```

If default is not given and key is not in the dictionary, a `KeyError` is raised.

```
>>> s = StackedDict()
>>> s.pop('a')
Traceback (most recent call last):
...
KeyError: 'a'
```

Affects only current layer.

```
>>> s = StackedDict(a=1, b=2, c=3)
>>> s.commit()
<wardrobe.stackdict.StackedDict object at 0x...>
>>> s.pop('b')
2
>>> silent = s.reset()
>>> s['b']
2
```

### **popitem()**

Remove and return some (key, value) pair as a 2-tuple.

```
>>> s = StackedDict(a=1)
>>> s.popitem()
('a', 1)
>>> len(s)
0
```

Raises `KeyError` if `D` is empty.

```
>>> s = StackedDict()
>>> s.popitem()
Traceback (most recent call last):
...
KeyError: 'popitem(): dictionary is empty'
```

Affects only the current layer.

```
>>> s = StackedDict(a=1)
>>> silent = s.commit()
>>> s.popitem()
('a', 1)
>>> silent = s.reset()
>>> dict(s)
{'a': 1}
```

### **reset()**

Restore dictionary to state before last `commit()`.

```
>>> s = StackedDict(a=1, b=2)
>>> s.commit().update(c=3, d=4)
>>> s.reset()
<wardrobe.stackdict.StackedDict object at 0x...>
>>> dict(s)
```

```
{'a': 1, 'b': 2}
>>> s.commit().update(a='A', b='B')
>>> s.reset()
<wardrobe.stackeddict.StackedDict object at 0x...>
>>> dict(s)
{'a': 1, 'b': 2}
>>> s.commit().update(a='A', c=3)
>>> s.reset()
<wardrobe.stackeddict.StackedDict object at 0x...>
>>> dict(s)
{'a': 1, 'b': 2}
```

Raises `NoRevisionException` when invoked on a `StackedDict` instance that hasn't been pushed yet.

```
>>> s = StackedDict()
>>> s.reset()
Traceback (most recent call last):
...
NoRevisionException
```

### **setdefault** (*key, default=None*)

If *key* is in the dictionary, return its value. If not, insert *key* with a value of *default* and return *default*. *default* defaults to `None`.

```
>>> s = StackedDict()
>>> s.setdefault('a', 1)
1
>>> s.setdefault('a', 2)
1
>>> s.setdefault('b') is None
True
```

### **update** (*\*args, \*\*kwargs*)

Update instance from dict (positional argument) and/or iterable (keyword arguments).

Affects only current layer.

Positional argument can be a dict...

```
>>> s = StackedDict(a=1, b=2)
>>> s.update({'a': 'A', 'c': 3})
>>> dict(s) == {'a': 'A', 'b': 2, 'c': 3}
True
```

... or any object that can be converted to a dict.

```
>>> s = StackedDict(a=1, b=2)
>>> s.update(('a', 'A'), ('c', 3))
>>> dict(s) == {'a': 'A', 'b': 2, 'c': 3}
True
```

Also accepts input as keyword arguments.

```
>>> s = StackedDict(a=1, b=2)
>>> s.update(a='A', c=3)
>>> dict(s) == {'a': 'A', 'b': 2, 'c': 3}
True
```

A combination of positional and keyword arguments is accepted. The positional argument is handled as a dict.

```
>>> s = StackedDict(a=1, b=2)
>>> s.update({'a': 'A'}, c=3)
>>> dict(s) == {'a': 'A', 'b': 2, 'c': 3}
True
```

But only one positional argument is accepted. This mimics a limitation of the standard dict type.

```
>>> s = StackedDict(a=1, b=2)
>>> s.update({'a': 'A'}, {'c': 3})
Traceback (most recent call last):
...
TypeError: update expected at most 1 arguments, got 2
```

#### **values()**

Return a copy of the StackedDict's list of values.

```
>>> s = StackedDict(a=1, b=2)
>>> values = s.values()
>>> values.sort()
>>> values
[1, 2]
```

#### **viewitems()**

Return a new view of the StackedDict's items ((key, value) pairs).

See <http://docs.python.org/library/stdtypes.html#dictionary-view-objects> for documentation of view objects.

```
>>> s = StackedDict()
>>> view = s.viewitems()
>>> view
dict_items([])
>>> s.update(a=1)
>>> view
dict_items([('a', 1)])
>>> s.commit().update(a='A')
>>> view
dict_items([('a', 'A')])
```

#### **viewkeys()**

See <http://docs.python.org/library/stdtypes.html#dictionary-view-objects> for documentation of view objects.

```
>>> s = StackedDict()
>>> view = s.viewkeys()
>>> view
dict_keys([])
>>> s.update(a=1)
>>> view
dict_keys(['a'])
>>> s.commit()
<wardrobe.stackeddict.StackedDict object at 0x...>
>>> del s['a']
>>> s['b'] = 2
>>> view
dict_keys(['b'])
```

#### **viewvalues()**

See <http://docs.python.org/library/stdtypes.html#dictionary-view-objects> for documentation of view objects.

```

>>> s = StackedDict()
>>> view = s.viewvalues()
>>> view
dict_values([])
>>> s.update(a=1)
>>> view
dict_values([1])
>>> s.commit()
<wardrobe.stackeddict.StackedDict object at 0x...>
>>> del s['a']
>>> s['b'] = 2
>>> view
dict_values([2])

```

### 1.2.3 wardrobe.exceptions

## 1.3 About wardrobe

This section is about the project itself.

### 1.3.1 Vision

wardrobe is about contextual tools for Python. Currently, it provides only one datastructure: `StackedDict`.

The project may provide more utilities or datastructures in future versions. But `StackedDict` may also become a standalone library.

### 1.3.2 Alternatives and related projects

This document presents other projects that provide similar or complementary functionalities. It focuses on differences with wardrobe.

#### Django's template contexts

Django's template contexts (`django.template.context.Context`) are dictionary-like objects that support `push()` and `pop()` methods. They are used to backup and restore the context in some template tags.

Some notes:

- wardrobe focuses on stack-based datastructures, whereas Django is a web framework.
- wardrobe is lighter than Django.
- `wardrobe.StackedDict` targets general Python usage, whereas Django's Context objects are specialized for use in the Django's template language.
- As of Django 1.4, `wardrobe.StackedDict` instances looks more like standard dict objects than Django's Context instances. As examples, look at `__delitem__()` or `pop()` methods.
- Django could use `wardrobe.StackedDict...` but, as of Django 1.4, it doesn't match the "almost no external dependencies" policy in Django project.

## Contextvars

`Contextvars` is about contextual variables. Not about contextual dictionary-like objects.

## References

### 1.3.3 License

Copyright (c) 2012, Benoît Bryon. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of wardrobe nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### 1.3.4 Authors & contributors

- Benoit Bryon <[benoit@marmelune.net](mailto:benoit@marmelune.net)>

### 1.3.5 Changelog

#### 0.1 (2012-07-26)

- Initial implementation of StackedDict: mimics standard dict class.

### 1.3.6 Why “wardrobe” name?

Because, when one gets dressed, he pushes layers of clothes, then poppes them.

## 1.4 Contributing to the project

This document provides guidelines for people who want to contribute to the wardrobe project.



### 1.4.1 Create tickets

Please use the [bugtracker](#)<sup>1</sup> **before** starting some work:

- check if the bug or feature request has already been filed. It may have been answered too!
- else create a new ticket.
- if you plan to contribute, tell us, so that we are given an opportunity to give feedback as soon as possible.
- Then, in your commit messages, reference the ticket with some `refs #TICKET-ID` syntax.

### 1.4.2 Fork and branch

- Work in forks and branches.
- Prefix your branch with the ticket ID corresponding to the issue. As an example, if you are working on ticket #23 which is about contribute documentation, name your branch like `23-contribute-doc`.

### 1.4.3 Setup a development environment

System requirements:

- [Python](#)<sup>2</sup> version 2.6 or 2.7, available as `python` command.

---

**Note:** You may use [Virtualenv](#)<sup>3</sup> to make sure the active `python` is the right one.

---

- `make` and `wget` to use the provided `Makefile`.

Execute:

```
git clone git@github.com:benoitbryon/wardrobe.git
cd wardrobe/
make develop
```

If you cannot execute the `Makefile`, read it and adapt the few commands it contains to your needs.

### 1.4.4 The Makefile

A `Makefile` is provided to ease development. Use it to:

- setup the development environment: `make develop`
- update it, as an example, after a pull: `make update`
- run tests: `make test`
- run benchmarks: `make benchmark`
- build documentation: `make documentation readme`

The `Makefile` is intended to be a live reference for the development environment.

---

<sup>1</sup> <https://github.com/benoitbryon/wardrobe/issues>

<sup>2</sup> <http://python.org>

<sup>3</sup> <http://virtualenv.org>

### 1.4.5 Documentation

Follow [style guide for Sphinx-based documentations](#) <sup>4</sup> when editing the documentation.

### 1.4.6 Test and build

Tests and builds will automatically be triggered before commit:

- tests include the build of documentation and README as HTML.
- a [Git pre-commit hook](#) <sup>5</sup> is installed during the development environment setup.

If you want to run them manually, use *the Makefile*.

### 1.4.7 References

---

<sup>4</sup> <http://documentation-style-guide-sphinx.readthedocs.org/>

<sup>5</sup> <http://git-scm.com/book/en/Customizing-Git-Git-Hooks>

---

### Ressources

---

- [online documentation](#)
- [PyPI page](#)
- [code repository](#)
- [bugtracker](#)



---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



## W

`wardrobe`, [3](#)

`wardrobe.stackeddict`, [4](#)





## A

args (wardrobe.stackeddickt.NoRevisionException attribute), 4

## C

clear() (wardrobe.stackeddickt.StackedDict method), 4  
commit() (wardrobe.stackeddickt.StackedDict method), 5  
copy() (wardrobe.stackeddickt.StackedDict method), 5

## F

fromkeys() (wardrobe.stackeddickt.StackedDict class method), 5

## G

get() (wardrobe.stackeddickt.StackedDict method), 5

## H

has\_key() (wardrobe.stackeddickt.StackedDict method), 6

## I

items() (wardrobe.stackeddickt.StackedDict method), 6  
iteritems() (wardrobe.stackeddickt.StackedDict method), 6  
iterkeys() (wardrobe.stackeddickt.StackedDict method), 7  
itervalues() (wardrobe.stackeddickt.StackedDict method), 7

## K

keys() (wardrobe.stackeddickt.StackedDict method), 7

## M

message (wardrobe.stackeddickt.NoRevisionException attribute), 4

## N

NoRevisionException, 4

## P

package\_dir (in module wardrobe), 3  
pop() (wardrobe.stackeddickt.StackedDict method), 7

popitem() (wardrobe.stackeddickt.StackedDict method), 8  
Python Enhancement Proposals  
PEP 396, 4

## R

reset() (wardrobe.stackeddickt.StackedDict method), 8

## S

setdefault() (wardrobe.stackeddickt.StackedDict method), 9  
StackedDict (class in wardrobe.stackeddickt), 4

## U

update() (wardrobe.stackeddickt.StackedDict method), 9

## V

values() (wardrobe.stackeddickt.StackedDict method), 10  
viewitems() (wardrobe.stackeddickt.StackedDict method), 10  
viewkeys() (wardrobe.stackeddickt.StackedDict method), 10  
viewvalues() (wardrobe.stackeddickt.StackedDict method), 10

## W

wardrobe (module), 3  
wardrobe.stackeddickt (module), 4