# baseline Documentation

*Release 1.2.1*

**Dan Gass**

**Dec 26, 2020**

# Contents

This tool streamlines creation and maintenance of tests which compare string output against a baseline. It offers a mechanism to compare a string against a baselined copy and update the baselined copy to match the new value when a mismatch occurs. The update process includes a manual step to facilitate a review of the change before acceptance. The tool uses multi-line string format for string baselines to improve readability for human review.

# CHAPTER 1

# Quick Start

Create an empty baseline with a triple quoted multi-line string. Place the ending triple quote on a separate line and indent it to the level you wish the string baseline update to be indented to. Add a compare of the string being tested to the baseline string. Then save the file as `fox.py`:

```python
from baseline import Baseline

expected = Baseline("""
    """)

test_string = """THE QUICK BROWN FOX
    JUMPS
OVER THE LAZY DOG."""

assert test_string == expected
```

Run `fox.py` and observe that the `assert` raises an exception since the strings are not equal. Because the comparison failed, the tool located the triple quoted baseline string in the source file and updated it with the mis-compared value. When the interpreter exited, the tool saved the updated source file but changed the file name to `fox.py.update`:

```python
from baseline import Baseline

expected = Baseline("""
    THE QUICK BROWN FOX
        JUMPS
    OVER THE LAZY DOG.
    """)

test_string = """THE QUICK BROWN FOX
    JUMPS
OVER THE LAZY DOG."""

assert test_string == expected
```

After reviewing the change with your favorite file differencing tool, accept the change by either manually overwriting the original file or use the `baseline` command line tool to scan the directory for updated scripts:

```
$ python -m baseline *
Found baseline updates for:
  fox.py

Hit [ENTER] to accept, [Ctrl-C] to cancel
```

Pressing `Enter` causes the tool to overwrite the scripts with the new baseline updates and remove the temporary `.py.update` files.

Run `fox.py` again and observe the `assert` does not raise an exception nor is a copy of the source file update generated. If in the future the test value changes, the `assert` raises an exception and causes a new source file update to be generated. Simply repeat the review and acceptance step and you are back in business!

## 1.1 [baseline] About

### 1.1.1 Contributors

- **Dan Gass (dan.gass at gmail dot com)**
  - Primary author
- **Adam Karpierz (akarpierz at gmail dot com)**
  - Python 3 support
  - Packaging
- **Peter Gessler (gessler.pd at gmail dot com)**
  - Travis C/I Setup

### 1.1.2 Development

**Repository** https://github.com/dmgass/baseline

### 1.1.3 License

(continues on next page)

(continued from previous page)

## 1.2 [baseline] API Reference

### 1.2.1 Classes

**class** baseline.**Baseline**

Baseline string.

Support comparison of a string against this baseline. When the comparison results in a mismatch, make a copy of the Python script containing the baseline and modify the baseline to match the new value.

### 1.2.2 Command Line

```
$ python -m baseline --help
usage: baseline [-h] [-w] [path [path ...]]

Locate scripts with baseline updates within the paths specified and modify the
scripts with the updates found. (The scripts to be modified will be summarized
and you will be offered a chance to cancel before files are changed.)

positional arguments:
  path        module or directory path

optional arguments:
  -h, --help  show this help message and exit
  -w, --walk  recursively walk directories
```

## 1.3 [baseline] Installation

### 1.3.1 Prerequisites

- Python
  - version 2.7
  - version 3.4 or higher
- Requirements for Installing Packages (located in the Installing Packages tutorial within the Python Packaging User Guide).

### 1.3.2 Install Steps

At a shell prompt, use pip to automatically download and install *baseline*:

```
python -m pip install --upgrade baseline
```

## 1.4 [baseline] Release Notes

Versions increment per semver.

**Note:** Changes to experimental features only result in a bump of the subminor (patch) version, including those introducing backwards incompatibility.

### 1.4.1 1.2.1 2020-DEC-26

- Fix `baseline` command line interface support for specifying directories to search. Previously, baseline inadvertently raised an exception if a directory other than the default (`'.'`) was specified.

### 1.4.2 1.2.0 2020-DEC-22

- Add `--force` command line option to suppress acknowledgement prompt.
- Add Python 3.9 support advertisement. (Regression testing added to release process.)
- Remove Python 3.4 and 3.5 support advertisement. (Regression testing removed from release process.) Nothing blocks installation, but no promise exists that the package works with those interpreter versions.

### 1.4.3 1.1.2 2020-MAY-02

- Maintain file permissions and owner (both when generating update file and when applying update file to original script). Previously, file owner and permissions were set based on permission levels of execution context which caused issues when executing under elevated permission levels (e.g. sudo).

### 1.4.4 1.1.1 2020-MAY-02

- Improve experimental feature to support specifying an alternative location. Add `BASELINE_MOVE_UPDATES` environment variable that when set to `YES`, enables specifying an alternative location to write update files. This master switch facilitates allowing CI/CD systems to enable the feature while leaving the feature off in local development while still using CI/CD resources (e.g. tox).

### 1.4.5 1.1.0 2020-MAY-01

- Add `--clean` (*-c*) option to baseline command line tool to remove update files.
- Add `--diff` (*-d*) option to baseline command line tool that shows difference and queries for overwrite permission for each updated file.

- Change update file extension to `.py.update` so that testing frameworks such as `unittest` or `pytest` ignore them.
- Add experimental feature to support specifying an alternative location to write update files with the `BASELINE_UPDATES_PATH` environment variable. (Note, `BASELINE_RELPATH_BASE` must be set when using this feature.)
- Add experimental feature to print contextual differences whenever a baseline mis-compare occurs. The feature may be turned on by setting the environment variable (`BASELINE_PRINT_DIFFS="YES"`) or overriding the class attribute (`Baseline.PRINT_DIFFS = True`).

### 1.4.6 1.0.0 2020-MAR-19

- Improve baseline update when multiple values compared against the same baseline. Generate a single multi-line baseline with headers between the various alternative values. This facilitates updating the baseline again.
- Support Python 3.8. Previously, when run using 3.8, the baseline update tool misplaced baseline updates in the first triple quoted string found above the baseline. (Python 3.8 stack frames now report the line number of the first line in a statement rather than the last.)
- Change behavior of `Baseline` to use raw strings when updating baselines when possible and improves readability.
- Deprecate `RawBaseline` since `Baseline` now incorporates its behavior.

### 1.4.7 Beta Releases

- **0.2.1 (2018-05-19)**
  - Fix command line tool to not raise UnboundedLocalError exception. Previously when tool was invoked with a wild card that yielded no baseline updates to move, an exception was unexpectedly raised.
- **0.2.0 (2018-05-18)**
  - Add `--movepath` command line option to move updated scripts to a new location instead of overwriting the original script (for use in continuous integration systems performing regression tests).
- **0.1.3 (2018-03-29)**
  - Show command line help dump in API reference documentation.
  - Fix development status classifier in setup configuration (to make PyPi listing accurate).
- **0.1.2 (2018-03-27)**
  - Add Travis C/I support.
  - Change author name to commonly used form.
  - Use Python 3.5 in tox for basic tasks.
  - Remove “beta” label.
- **0.1.1 (2018-03-25)**
  - Change author to match PyPi user name.
- **0.1.0 (2018-03-25)**
  - Initial “beta” release.

## 1.5 [baseline] Usage

### 1.5.1 One line Strings

To create a baseline string that contains a special update mechanism, use triple quotes around the string and instantiate `Baseline` with it. The resulting Python string object supports natural equality comparisons:

Listing 1: hello.py

```
from baseline import Baseline

expected = Baseline("""Hello""")

test_string = "Hello World!"

assert test_string == expected
```

Run the script and observe that the `assert` raises an exception since the strings are not equal. Because the comparison failed, the tool located the triple quoted baseline string in the source file and updated it with the miscompared value. When the interpretter exited, the tool saved the updated source file using the file extension `.py.update`):

Listing 2: hello.py.update

```
expected = Baseline("""Hello World!""")

test_string = "Hello World!"

assert test_string == expected
```

After reviewing the change with your favorite file differencing tool, accept the change by either manually overwriting the original file or use the `baseline` command line interface to scan the directory for baseline update files:

```
$ python -m baseline *
Found baseline updates for:
  hello.py

Hit [ENTER] to update, [Ctrl-C] to cancel
```

Pressing *Enter* causes the tool to overwrite the scripts with the new baseline updates and remove the temporary *.py.update* files.

### 1.5.2 Multi-Line Strings

The triple quote usage in the `Baseline` instantiation provides a consistent search and replace mechanism that supports embedding quotation marks and newlines within a baselined string. Embedding newlines improves the strings human readability which makes reviewing updates easier.

For multiline baselined string format, start the string on the line following the opening triple quote delimiter. Insert a line after the baselined string content to hold the closing triple quote delimiter. Indent the closing triple quote delimiter to the indentation level of the baselined string:

```
from baseline import Baseline

expected = Baseline("""
    THE QUICK BROWN FOX
        JUMPS
    OVER THE LAZY DOG.
    """)

test_string = "THE QUICK BROWN FOX\n    JUMPS\nOVER THE LAZY DOG."

assert test_string == expected
```

The example above executes without an assertion because the tool strips the leading indentation of every line in the baselined string based on the indentation of the closing triple quote.

### 1.5.3 Transforms

Often strings to test against a baseline contain substrings that may vary from one execution to the next. Before the comparison, normalize the string by substituting a representative constant value. For example, use a regular expression to transform a variable time into a constant value:

```
import re
import time

from baseline import Baseline


expected = Baseline("""The time is HH:MM:SS.""")

test_string = "The time is {}.".format(time.strftime("%H:%M:%S"))

assert re.sub(r'\d\d:\d\d:\d\d', 'HH:MM:SS', test_string) == expected
```

If this is a common operation or there are multiple transformations needed, override the `TRANSFORMS` class attribute and list the operations to be performed. The tool performs each of the operations on the test string before every comparison.

```
import re
import time

from baseline import Baseline


def normalize_time(s):
    return re.sub(r'\d\d:\d\d:\d\d', 'HH:MM:SS', s)

```

(continues on next page)

(continued from previous page)

```
class NormalizedBaseline(Baseline):

    """Normalized string baseline."""

    TRANSFORMS = [normalize_time]


expected = NormalizedBaseline("""The time is HH:MM:SS.""")

test_string = "The time is {}.".format(time.strftime("%H:%M:%S"))

assert test_string == expected
```

### 1.5.4 Tips and Tricks

#### Quick Tips

- Take your time and be diligent in your review of baseline updates. Similar to Python itself, this tool provides a lot of rope, don't hang yourself.
- Put comments above the baseline to provide information to a future maintainer of the important aspects of the baseline that are the focus of the test.
- Feel free to baseline strings with any style triple quotes embedded. The tool adjusts and uses the alternative style. If the test string contains both styles, transform one style into something else before comparison.
- To archive resulting test script updates from a regression test run within a continuous integration system, use the `--movepath` command line option to move updated scripts to a new location instead of overwriting the original script. T

#### Initial Baseline Value

To avoid the work of anticipating the exact content of the string baseline, specify an empty baseline in multi-line format and set the indentation level with the closing triple quote:

```
from baseline import Baseline

expected = Baseline("""
    """)

test_string = "THE QUICK BROWN FOX\n  JUMPS\nOVER THE LAZY DOG."

assert test_string == expected
```

Run the script and let the tool fill in the string baseline. Then carefully review the baseline update and accept.

# Index