Hunter

Release 0.6.0

December 23, 2015

Contents

Contents:

CHAPTER 1

Overview

1.1 Hunter

docs	
tests	
package	

Hunter is a flexible code tracing toolkit, not for measuring coverage, but for debugging, logging, inspection and other nefarious purposes. It has a simple Python API and a convenient terminal API (see *Environment variable activation*).

API is considered unstable until 1.0 is released.

- Free software: BSD license

1.1.1 Installation

```
pip install hunter
```

1.1.2 Documentation

https://python-hunter.readthedocs.org/

1.1.3 Overview

The default action is to just print the code being executed. Example:

```
import hunter
hunter.trace(module='posixpath')

import os
os.path.join('a', 'b')
```

Would result in:

```
python2.7/posixpath.py:60    call      def join(a, *p):
python2.7/posixpath.py:64    line          path = a
python2.7/posixpath.py:65    line          for b in p:
python2.7/posixpath.py:66    line              if b.startswith('/'):
python2.7/posixpath.py:68    line              elif path == '' or path.endswith('/'):
python2.7/posixpath.py:71    line                  path += '/' + b
python2.7/posixpath.py:65    line          for b in p:
python2.7/posixpath.py:72    line          return path
python2.7/posixpath.py:72    return        return path
                             ...       return value: 'a/b'
```

- or in a terminal:

You can have custom actions, like a variable printer - example:

```
import hunter
hunter.trace(hunter.Q(module='posixpath', action=hunter.VarsPrinter('path')))

import os
os.path.join('a', 'b')
```

Would result in:

```
python2.7/posixpath.py:60    call      def join(a, *p):
python2.7/posixpath.py:64    line          path = a
                             vars      path => 'a'
python2.7/posixpath.py:65    line          for b in p:
                             vars      path => 'a'
python2.7/posixpath.py:66    line              if b.startswith('/'):
                             vars      path => 'a'
python2.7/posixpath.py:68    line              elif path == '' or path.endswith('/'):
                             vars      path => 'a'
python2.7/posixpath.py:71    line                  path += '/' + b
                             vars      path => 'a/b'
python2.7/posixpath.py:65    line          for b in p:
                             vars      path => 'a/b'
python2.7/posixpath.py:72    line          return path
                             vars      path => 'a/b'
python2.7/posixpath.py:72    return        return path
                             ...       return value: 'a/b'
```

- or in a terminal:

You can give it a tree-like configuration where you can optionally configure specific actions for parts of the tree (like dumping variables or a pdb set_trace):

TODO: More examples.

Environment variable activation

For your convenience environment variable activation is available. Just run your app like this:

```
PYTHONHUNTER="module='os.path'" python yourapp.py
```

On Windows you'd do something like:

```
set PYTHONHUNTER=module='os.path'
python yourapp.py
```

The activation works with a clever `.pth` file that checks for that env var presence and before your app runs does something like this:

```
from hunter import *
trace(<whatever-you-had-in-the-PYTHONHUNTER-env-var>)
```

That also means that it will do activation even if the env var is empty, eg: `PYTHONHUNTER=""`.

1.1.4 Development

To run the all tests run:

```
tox
```

CHAPTER 2

Installation

At the command line:

```
pip install hunter
```

CHAPTER 3

Introduction

3.1 Installation

To install hunter run:

```
pip install hunter
```

3.2 The `trace` function

The *hunter.trace* function can take 2 types of arguments:

- Keyword arguments like `module`, `function` or `action`. This is for convenience.
- Callbacks that take an `event` argument:
  - Builtin predicates like: *hunter.Query*, *hunter.When*, *hunter.And* or *hunter.Or*.
  - Actions like: *hunter.CodePrinter*, *hunter.Debugger* or *hunter.VarsPrinter*

Note that *hunter.trace* will use *hunter.Q* when you pass multiple positional arguments or keyword arguments.

3.3 The `Q` function

The *hunter.Q* function provides a convenience API for you:

- `Q(module='foobar')` is converted to `Query(module='foobar')`.
- `Q(module='foobar', action=Debugger)` is converted to `When(Query(module='foobar'), Debugger)`.
- `Q(module='foobar', actions=[CodePrinter, VarsPrinter('name')])` is converted to `When(Query(module='foobar'), CodePrinter, VarsPrinter('name'))`.
- `Q(Q(module='foo'), Q(module='bar'))` is converted to `Or(Q(module='foo'), Q(module='bar'))`.
- `Q(your_own_callback, module='foo')` is converted to `Or(your_own_callback, Q(module='foo'))`.

Note that the default junction *hunter.Q* uses is *hunter.Or*.

3.4 The builtin predicates and actions

All the builtin predicates (*hunter.Query*, *hunter.When*, *hunter.And* and *hunter.Or*) support the `|` and `&` operators:

- `Query(module='foo') | Query(module='bar')` is converted to `Or(Query(module='foo'), Query(module='bar'))`
- `Query(module='foo') & Query(module='bar')` is converted to `And(Query(module='foo'), Query(module='bar'))`

3.5 Activation

You can activate Hunter in two ways.

3.5.1 *via* code

```
import hunter
hunter.trace(
    ...
)
```

3.5.2 *via* environment variable

Set the `PYTHONHUNTER` environment variable. Eg:

```
PYTHONHUNTER="module='os.path'" python yourapp.py
```

On Windows you'd do something like:

```
set PYTHONHUNTER=module='os.path'
python yourapp.py
```

The activation works with a clever `.pth` file that checks for that env var presence and before your app runs does something like this:

```
from hunter import *
trace(
    <whatever-you-had-in-the-PYTHONHUNTER-env-var>
)
```

That also means that it will do activation even if the env var is empty, eg: `PYTHONHUNTER=""`.

CHAPTER 4

Cookbook

TODO

CHAPTER 5

Reference

5.1 hunter

Functions

hunter.trace	Starts tracing.
hunter.stop	Stop tracing.
hunter.Q	Handles situations where *hunter.Query* objects (or other callables) are passed in as positional arguments.

Predicates

hunter.Query	A query class.
hunter.When	Runs `actions` when `condition(event)` is `True`.
hunter.And	*And* predicate. Exits at the first sub-predicate that returns `False`.
hunter.Or	*Or* predicate. Exits at first sub-predicate that returns `True`.

Actions

hunter.CodePrinter	An action that just prints the code being executed.
hunter.Debugger	An action that starts `pdb`.
hunter.VarsPrinter	An action that prints local variables and optionally global variables visible from the current executing fr

Objects

hunter.Event	Event wrapper for `frame, kind, arg` (the arguments the settrace function gets).

hunter.**trace**(**predicates*, ***options*)

Starts tracing. Can be used as a context manager (with slightly incorrect semantics - it starts tracing before `__enter__` is called).

Parameters

- **predicates** (*hunter.Q* instances) – Runs actions if any of the given predicates match.

- **options** – Keyword arguments that are passed to *hunter.Q*, for convenience.

hunter.**stop**()
Stop tracing. Restores previous tracer (if any).

class hunter.**Q**
Handles situations where *hunter.Query* objects (or other callables) are passed in as positional arguments. Conveniently converts that to an *hunter.Or* predicate.

class hunter.**Query**(***query*)
A query class.

See *hunter.Event* for fields that can be filtered on.

__and__(*other*)
Convenience API so you can do `Q() & Q()`. It converts that to `And(Q(), Q())`.

__call__(*event*)
Handles event. Returns True if all criteria matched.

__init__(***query*)

Parameters **query** – criteria to match on.

Accepted arguments: `arg`, `code`, `filename`, `frame`, `fullsource`, `function`, `globals`, `kind`, `lineno`, `locals`, `module`, `source`, `stdlib`, `tracer`.

__or__(*other*)
Convenience API so you can do `Q() | Q()`. It converts that to `Or(Q(), Q())`.

class hunter.**When**(*condition*, **actions*)
Runs `actions` when `condition(event)` is `True`.

Actions take a single `event` argument.

__call__(*event*)
Handles the event.

class hunter.**And**(**predicates*)
And predicate. Exits at the first sub-predicate that returns `False`.

__call__(*event*)
Handles the event.

class hunter.**Or**(**predicates*)
Or predicate. Exits at first sub-predicate that returns `True`.

__call__(*event*)
Handles the event.

class hunter.**CodePrinter**(*stream=<open file '<stderr>', mode 'w'>, force_colors=False, filename_alignment=40*)
An action that just prints the code being executed.

Parameters

- **stream** (*file-like*) – Stream to write to. Default: `sys.stderr`.
- **filename_alignment** (*int*) – Default size for the filename column (files are right-aligned). Default: `40`.

__call__(*event*, *sep='/'*, *join=<function join>*)
Handle event and print filename, line number and source code. If event.kind is a *return* or *exception* also prints values.

class hunter.**Debugger**(*klass=<class pdb.Pdb>*, ***kwargs*)
An action that starts pdb.

__call__(*event*)
Runs a pdb.set_trace at the matching frame.

class hunter.**VarsPrinter**(**names*, ***options*)
An action that prints local variables and optionally global variables visible from the current executing frame.

Parameters

- ***names** (*strings*) – Names to evaluate. Expressions can be used (will only try to evaluate if all the variables are present on the frame.
- **stream** (*file-like*) – Stream to write to. Default: sys.stderr.
- **filename_alignment** (*int*) – Default size for the filaneme column (files are right-aligned). Default: 40.
- **globals** (*bool*) – Allow access to globals. Default: False (only looks at locals).

__call__(*event*)
Handle event and print the specified variables.

static _iter_symbols(*code*)
Iterate all the variable names in the given expression.

Example:

- self.foobar yields self
- self[foobar] yields *self*‘ and foobar

_safe_eval(*code*, *event*)
Try to evaluate the given code on the given frame. If failure occurs, returns some ugly string with exception.

class hunter.**Event**(*frame*, *kind*, *arg*, *tracer*)
Event wrapper for frame, kind, arg (the arguments the settrace function gets).

Provides few convenience properties.

code
A code object (not a string).

filename
A string with absolute path to file.

fullsource
A string with the sourcecode for the current statement (from linecache - failures are ignored).

May include multiple lines if it’s a class/function definition (will include decorators).

function
A string with function name.

globals
A dict with global variables.

lineno
An integer with line number in file.

locals
A dict with local variables.

module
A string with module name (eg – "foo.bar").

source
A string with the sourcecode for the current line (from `linecache` - failures are ignored).

Fast but sometimes incomplete.

stdlib
A boolean flag. `True` if frame is in stdlib.

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

6.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.2 Documentation improvements

Hunter could always use more documentation, whether as part of the official Hunter docs, in docstrings, or even on the web in blog posts, articles, and such.

6.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/ionelmc/python-hunter/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.4 Development

To set up *python-hunter* for local development:

1. Fork python-hunter on GitHub.
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/python-hunter.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

6.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`) [1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

6.4.2 Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 7

Authors

- Ionel Cristian Mărie - http://blog.ionelmc.ro

CHAPTER 8

Changelog

8.1 0.6.0 (2015-10-10)

- Added a `clear_env_var` option on the tracer (disables tracing in subprocess).
- Added `force_colors` option on `VarsPrinter` and `CodePrinter`.
- Allowed setting the *stream* to a file name (option on `VarsPrinter` and `CodePrinter`).
- Bumped up the filename alignment to 40 cols.
- If not merging then *self* is not kept as a previous tracer anymore. Closes #16.
- Fixed handling in VarsPrinter: properly print eval errors and don't try to show anything if there's an AttributeError. Closes #18.
- Added a `stdlib` boolean flag (for filtering purposes). Closes #15.
- Fixed broken frames that have "None" for filename or module (so they can still be treated as strings).
- Corrected output files in the `install_lib` command so that pip can uninstall the pth file. This only works when it's installed with pip (sadly, `setup.py install/develop` and `pip install -e` will still leave pth garbage on `pip uninstall hunter`).

8.2 0.5.1 (2015-04-15)

- Fixed `Event.globals` to actually be the dict of global vars (it was just the locals).

8.3 0.5.0 (2015-04-06)

- Fixed `And` and `Or` "single argument unwrapping".
- Implemented predicate compression. Example: `Or(Or(a, b), c)` is converted to `Or(a, b, c)`.
- Renamed the `Event.source` to `Event.fullsource`.
- Added `Event.source` that doesn't do any fancy sourcecode tokenization.
- Fixed `Event.fullsource` return value for situations where the tokenizer would fail.
- Made the print function available in the `PYTHONHUNTER` env var payload.
- Added a __repr__ for `Event`.

8.4 0.4.0 (2015-03-29)

- Disabled colors for Jython (contributed by Claudiu Popa in #12).
- Test suite fixes for Windows (contributed by Claudiu Popa in #11).
- Added an introduction section in the docs.
- Implemented a prettier fallback for when no sources are available for that frame.
- Implemented fixups in cases where you use action classes as a predicates.

8.5 0.3.1 (2015-03-29)

- Forgot to merge some commits ...

8.6 0.3.0 (2015-03-29)

- Added handling for internal repr failures.
- Fixed issues with displaying code that has non-ascii characters.
- Implemented better display for `call` frames so that when a function has decorators the function definition is shown (instead of just the first decorator). See: #8.

8.7 0.2.1 (2015-03-28)

- Added missing color entry for exception events.
- Added `Event.line` property. It returns the source code for the line being run.

8.8 0.2.0 (2015-03-27)

- Added color support (and `colorama` as dependency).
- Added support for expressions in `VarsPrinter`.
- Breaking changes:
  - Renamed `F` to `Q`. And `Q` is now just a convenience wrapper for `Query`.
  - Renamed the `PYTHON_HUNTER` env variable to `PYTHONHUNTER`.
  - Changed `When` to take positional arguments.
  - Changed output to show 2 path components (still not configurable).
  - Changed `VarsPrinter` to take positional arguments for the names.
- Improved error reporting for env variable activation (`PYTHONHUNTER`).
- Fixed env var activator (the `.pth` file) installation with `setup.py install` (the "egg installs") and `setup.py develop`/`pip install -e` (the "egg links").

8.9 0.1.0 (2015-03-22)

- First release on PyPI.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

h

Symbols

A

C

D

E

F

G

H

L

M

O

Q

S

T

V

W