remote-pdb

Release 2.1.0

Jul 24, 2020

Contents

CHAPTER 1

Overview

docs	
tests	
package	

Remote vanilla PDB (over TCP sockets) *done right*: no extras, proper handling around connection failures and CI. Based on pdbx.

- Free software: BSD 2-Clause License

1.1 Installation

```
pip install remote-pdb
```

1.2 Usage

To open a remote PDB on first available port:

```
from remote_pdb import set_trace
set_trace() # you'll see the port number in the logs
```

To use some specific host/port:

```
from remote_pdb import RemotePdb
RemotePdb('127.0.0.1', 4444).set_trace()
```

To connect just run `telnet 127.0.0.1 4444`. When you are finished debugging, either exit the debugger, or press Control-], then Control-d.

Alternately, one can connect with NetCat: `nc -C 127.0.0.1 4444` or Socat: `socat readline tcp:127.0.0.1:4444` (for line editing and history support). When finished debugging, either exit the debugger, or press Control-c.

Note that newer Ubuntu disabled readline support in socat, so if you get `unknown device/address "readline"` try using rlwrap like this:

```
rlwrap socat - tcp:127.0.0.1:4444
```

1.3 Using in containers

If you want to connect from the host to remote-pdb running inside the container you should make sure that:

- The port you will use is mapped (eg: `-p 4444:4444`).
- The host is set to `0.0.0.0` (`localhost` or `127.0.0.1` will not work because Docker doesn't map the port on the local interface).

1.4 Integration with breakpoint() in Python 3.7+

If you are using Python 3.7 one can use the new `breakpoint()` built in to invoke remote PDB. In this case the following environment variable must be set:

```
PYTHONBREAKPOINT=remote_pdb.set_trace
```

The debugger can then be invoked as follows, without any imports:

```
breakpoint()
```

As the `breakpoint()` function does not take any arguments, environment variables can be used to specify the host and port that the server should listen to. For example, to run `script.py` in such a way as to make `telnet 127.0.0.1 4444` the correct way of connecting, one would run:

```
PYTHONBREAKPOINT=remote_pdb.set_trace REMOTE_PDB_HOST=127.0.0.1 REMOTE_PDB_PORT=4444␣
↪python script.py
```

If `REMOTE_PDB_HOST` is omitted then a default value of 127.0.0.1 will be used. If `REMOTE_PDB_PORT` is omitted then the first available port will be used. The connection information will be logged to the console, as with calls to `remote_pdb.set_trace()`.

To quiet the output, set `REMOTE_PDB_QUIET=1`, this will prevent `RemotePdb` from producing any output – you'll probably want to specify `REMOTE_PDB_PORT` as well since the randomized port won't be printed.

1.5 Note about OS X

In certain scenarios (backgrounded processes) OS X will prevent readline to be imported (and readline is a dependency of pdb). A workaround (run this early):

```
import signal
signal.signal(signal.SIGTTOU, signal.SIG_IGN)
```

See #9 and cpython#14892.

1.6 Requirements

Python 2.6, 2.7, 3.2, 3.3 and PyPy are supported.

1.7 Similar projects

- qdb

CHAPTER 2

Installation

At the command line:

```
pip install remote-pdb
```

CHAPTER 3

Usage

To use remote-pdb in a project:

```
import remote_pdb
```

CHAPTER 4

Reference

4.1 remote_pdb

class remote_pdb.**RemotePdb**(*host*, *port*, *patch_stdstreams=False*, *quiet=False*)

This will run pdb as a ephemeral telnet service. Once you connect no one else can connect. On construction this object will block execution till a client has connected.

Based on https://github.com/tamentis/rpdb I think ...

To use this:

```
RemotePdb(host='0.0.0.0', port=4444).set_trace()
```

Then run: telnet 127.0.0.1 4444

do_exit(*arg*)

q(uit) exit

Quit from the debugger. The program being executed is aborted.

do_q(*arg*)

q(uit) exit

Quit from the debugger. The program being executed is aborted.

do_quit(*arg*)

q(uit) exit

Quit from the debugger. The program being executed is aborted.

set_trace(*frame=None*)

Start debugging from frame.

If frame is not specified, debugging starts from caller's frame.

remote_pdb.**set_trace**(*host=None*, *port=None*, *patch_stdstreams=False*, *quiet=None*)

Opens a remote PDB on first available port.

CHAPTER 5

Contributing

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

5.1 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.

5.2 Documentation improvements

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

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/ionelmc/python-remote-pdb/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 code contributions are welcome :)

5.4 Development

To set up *python-remote-pdb* for local development:

1. Fork python-remote-pdb (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:YOURGITHUBNAME/python-remote-pdb.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 and docs builder 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.

5.4.1 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`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -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 6

Authors

- Ionel Cristian Mărieș - https://blog.ionelmc.ro
- Matthew Wilkes - https://github.com/MatthewWilkes
- Anthony Sottile - https://github.com/asottile
- Terence Honles - https://github.com/terencehonles

CHAPTER 7

Changelog

7.1 2.1.0 (2020-07-24)

- Changed logging to use `remote_pdb` logger instead of the root one. Contributed by Terence Honles in #24.

7.2 2.0.0 (2019-07-31)

- Fixed inconsistency with normal use of `pdb` - *BdbQuit* will now be raised on quitting. Contributed by Anthony Sottile in #18. **BACKWARDS INCOMPATIBLE**.
- Added `REMOTE_PDB_QUIET=1` to silence output. Contributed by Anthony Sottile in #19.

7.3 1.3.0 (2019-03-13)

- Documented support for Python 3.7's `breakpoint()`.
- Added support for setting the socket listening host/port through the `REMOTE_PDB_HOST`/`REMOTE_PDB_PORT` environment variables. Contributed by Matthew Wilkes in #14.
- Removed use of *rw* file wrappers around sockets (turns out socket's `makefile` is very buggy in Python 3.6 and later - output is discarded). Contributed in #13.

7.4 1.2.0 (2015-09-26)

- Always print/log listening address.

7.5 1.1.3 (2015-07-06)

- Corrected the default frame tracing starts from.

7.6 1.1.2 (2015-07-06)

- Small readme update.

7.7 1.1.1 (2015-07-06)

- Remove bogus `remote_pdb` console script.

7.8 1.1.0 (2015-06-21)

- Fixed buffering issues when running on Python 3 and Windows.

7.9 1.0.0 (2015-06-15)

- Added support for PDB++.

7.10 0.2.1 (2014-03-07)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index