Writing an autoreloader in Python

EuroPython 2019

Tom Forbes - tom@tomforb.es
1. What is an autoreloader?

2. Django's implementation

3. Rebuilding it

4. The aftermath
What is an autoreloader?

A component in a larger system that detects and applies changes to source code, without developer interaction.
Hot reloader

A special type of autoreloader that reloads your changes without restarting the system.

Shout out to Erlang where you hot-reload code while deploying
But Python has `reload()`?

```python
import time
import my_custom_module

while True:
    time.sleep(1)
    reload(my_custom_module)
```
Dependencies are the enemy of a hot reloader

Python modules have *lots* of inter-dependencies
Imagine you wrote a hot-reloader for Python

You import a function inside your_module:

from another_module import some_function

Then you replace some_function with new code.

After reloading, what does your_module.some_function reference?
So how do we reload code in Python?
We turn it off and on again

HAVE YOU TRIED TURNING IT OFF AND ON AGAIN?
We restart the process.
On every code change.
Over and over again.
When you run `manage.py runserver`:

1. **Django re-executes** `manage.py runserver` with a specific environment variable set
2. The child process runs Django, and watches for any file changes
3. When a change is detected it exits with a specific exit code (3)
4. The parent Django process restarts it.
The history of the Django autoreloader

First commit in 2005

No major changes until 2013 when inotify support was added

kqueue support was also added in 2013, then removed 1 month later
Summary so far:

1. An autoreloader is a common development tool
2. Hot reloaders are really hard to write in Python
3. Python autoreloaders restart the process on code changes
4. The Django autoreloader was old and hard to extend
(Re-)Building an autoreloader

Three or four steps:

1. Find files to monitor
2. Wait for changes and trigger a reload
3. Make it testable
4. Bonus points: Make it efficient
Finding files to monitor

sys.modules

👉 ipython -c 'import sys; print(len(sys.modules))'
   642

👉 python -c 'import sys; print(len(sys.modules))'
   42
Finding files to monitor

Sometimes things that are *not* modules find their way inside `sys.modules`

```
ipython -c 'import sys; print(sys.modules["typing.io"])'
<class 'typing.io'>
```
Python's imports are very dynamic

The import system is unbelievably flexible

Can import from .zip files, or from .pyc files directly

https://github.com/nvbn/import_from_github_com

from github_com.kennethreitz import requests
What can you do?
Finding files: The simplest implementation

```python
import sys

def get_files_to_watch():
    return [
        module.__spec__.origin
        for module in sys.modules.values()
    ]
```
(Re-)Building an autoreloader

Three or four steps:

1. Find files to monitor
2. Wait for changes and trigger a reload
3. Make it testable
4. Bonus points: Make it efficient
Waiting for changes

All\(^1\) filesystems report the last modification of a file

\[
mtime = \text{os.stat}('/etc/password').st_mtime
\]

\[
\text{print}(mtime)
\]

1561338330.0561554

\(^1\)Except when they don't
Filesystems can be *weird*.

**HFS+**: 1 second time resolution

**Windows**: 100ms intervals (files may appear in the future 😱)

**Linux**: Depends on your hardware clock!

```python
p = pathlib.Path('test')
p.touch()
# 5 milliseconds
```

```python
time.sleep(0.005)  # 5 milliseconds
p.touch()
```
Filesystems can be *weird*.

Network filesystems mess things up completely

`os.stat()` *suddenly becomes expensive!*
import time, os

def watch_files():
    file_times = {}  # Maps paths to last modified times
    while True:
        for path in get_files_to_watch():
            mtime = os.stat(path).st_mtime
            previous_mtime = file_times.setdefault(path, mtime)
            if mtime != previous_mtime:
                exit(3)  # Change detected!
        time.sleep(1)
(Re-)Building an autoreloader

Three or four steps:

1. **Find files to monitor**
2. **Wait for changes and trigger a reload**
3. **Make it testable**
4. **Bonus points: Make it efficient**
Making it testable

Not many tests in the wider ecosystem

<table>
<thead>
<tr>
<th>Project</th>
<th>Test Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tornado</td>
<td>2</td>
</tr>
<tr>
<td>Flask</td>
<td>3</td>
</tr>
<tr>
<td>Pyramid</td>
<td>6</td>
</tr>
</tbody>
</table>
Making it testable

Reloaders are infinite loops that run in threads and rely on a big ball of external state.
Generators!
def watch_files(sleep_time=1):
    file_times = {}
    while True:
        for path in get_files_to_watch():
            mtime = os.stat(path).st_mtime
            previous_mtime = file_times.setdefault(path, mtime)
            if mtime > previous_mtime:
                exit(3)
            time.sleep(sleep_time)
    yield
def test_it_works(tmp_path):
    reloader = watch_files(sleep_time=0)
    next(reloader)  # Initial tick
    increment_file_mtime(tmp_path)
    with pytest.raises(SystemExit):
        next(reloader)
(Re-)Building an autoreloader

Three or four steps:

1. Find files to monitor
2. Wait for changes and trigger a reload
3. Make it testable
4. Bonus points: Make it efficient
Making it efficient

Slow parts:

1. Iterating modules
2. Checking for file modifications
Making it efficient: Iterating modules

```python
import sys, functools

def get_files_to_watch():
    return sys_modules_files(frozenset(sys.modules.values()))

@functools.lru_cache(maxsize=1)
def sys_modules_files(modules):
    return [module.__spec__.origin for module in modules]
```
Making it efficient: Skipping the stdlib + third party packages
Making it efficient: Skipping the stdlib + third party packages

```python
import site
site.getsitepackages()
```

Not available in a virtualenv 😱
Making it efficient: Skipping the stdlib + third party packages

```python
import distutils.sysconfig
print(distutils.sysconfig.get_python_lib())
```

Works, but some systems (Debian) have more than one site package directory.
Making it efficient: Skipping the stdlib + third party packages

It all boils down to:

Risk vs Reward
Making it efficient: Filesystem notifications
Making it efficient: Filesystem notifications

Each platform has different ways of handling this
Watchdog\(^2\) implements 5 different ways - 3,000 LOC!
They are all \textit{directory} based.

\(^2\)https://github.com/gorakhargosh/watchdog/tree/master/src/watchdog/observers
Making it efficient: Filesystem notifications

https://facebook.github.io/watchman/
Making it efficient: Filesystem notifications

```python
import watchman

def watch_files(sleep_time=1):
    server = watchman.connect_to_server()
    for path in get_files_to_watch():
        server.watch_file(path)
    while True:
        changes = server.wait(timeout=sleep_time)
        if changes:
            exit(3)
        yield
```

Tom Forbes - EuroPython 2019
(Re-)Building an autoreloader

Three or four steps:

1. Find files to monitor
2. Wait for changes and trigger a reload
3. Make it testable
4. Bonus points: Make it efficient
The aftermath

✔ Much more modern, easy to extend code
✔ Faster, and can use Watchman if available
✔ 72 tests 🎉
✔ No longer a "dark corner" of Django\(^3\)

\(^3\) I might be biased!

Tom Forbes - EuroPython 2019
# The aftermath

## #30554 closed Uncategorized (invalid)
**Excessive logging by autoreload in v 2.2.1 and 2.2.2**

<table>
<thead>
<tr>
<th>Reported by:</th>
<th>Phoebe Bright</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component:</td>
<td>Core (Other)</td>
</tr>
<tr>
<td>Severity:</td>
<td>Normal</td>
</tr>
<tr>
<td>Cc:</td>
<td>Tom Forbes</td>
</tr>
<tr>
<td>Owned by:</td>
<td>nobody</td>
</tr>
<tr>
<td>Version:</td>
<td>2.2</td>
</tr>
<tr>
<td>Keywords:</td>
<td></td>
</tr>
<tr>
<td>Triage Stage:</td>
<td>Unreviewed</td>
</tr>
</tbody>
</table>

Opened: 2 weeks ago  
Closed: 13 days ago  
Last modified: 13 days ago

## #30516 closed Bug (fixed)
**Autoreloader crashes on re-raising exceptions with custom signature.**

<table>
<thead>
<tr>
<th>Reported by:</th>
<th>Alan Trick</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component:</td>
<td>Utilities</td>
</tr>
<tr>
<td>Severity:</td>
<td>Release blocker</td>
</tr>
<tr>
<td>Cc:</td>
<td>Tom Forbes</td>
</tr>
<tr>
<td>Owned by:</td>
<td>Tom Forbes</td>
</tr>
<tr>
<td>Version:</td>
<td>2.2</td>
</tr>
<tr>
<td>Keywords:</td>
<td>autoreload</td>
</tr>
<tr>
<td>Triage Stage:</td>
<td>Accepted</td>
</tr>
</tbody>
</table>

Opened: 4 weeks ago  
Closed: 4 weeks ago  
Last modified: 4 weeks ago
The aftermath

#30506 closed Bug (needsinfo)

Auto-reloading with StatReloader very intermittently throws "ValueError: embedded null byte".

Reported by: Keryn Knight
Component: Core (Management commands)
Severity: Normal
Cc: Tom Forbes

Owned by: nobody
Version: master
Keywords:
Triage Stage: Unreviewed

#30479 closed Bug (fixed)

Autoreloader with StatReloader doesn't track changes in manage.py.

Reported by: Keryn Knight
Component: Utilities
Severity: Release blocker
Cc: Tom Forbes

Owned by: Tom Forbes
Version: 2.2
Keywords: autoreload
Triage Stage: Accepted
### The aftermath

#### #30372 closed Bug (needsinfo)

**Django (moderately) High CPU usage at Idle**

<table>
<thead>
<tr>
<th>Reported by:</th>
<th>Benjamin Schollnick</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component:</td>
<td>Core (Other)</td>
</tr>
<tr>
<td>Severity:</td>
<td>Normal</td>
</tr>
<tr>
<td>Cc:</td>
<td>Tobias Kunze, Tom Forbes</td>
</tr>
<tr>
<td>Owned by:</td>
<td>nobody</td>
</tr>
<tr>
<td>Version:</td>
<td>2.2</td>
</tr>
<tr>
<td>Keywords:</td>
<td></td>
</tr>
<tr>
<td>Triage Stage:</td>
<td>Unreviewed</td>
</tr>
</tbody>
</table>

Opened 2 months ago  
Closed 8 weeks ago  
Last modified 4 weeks ago

#### #30366 closed Bug (fixed)

**The StatReloaderTests will fail on Mac OSX when HFS+ is used as a filesystem**

<table>
<thead>
<tr>
<th>Reported by:</th>
<th>Martijn Jacobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component:</td>
<td>Testing framework</td>
</tr>
<tr>
<td>Severity:</td>
<td>Normal</td>
</tr>
<tr>
<td>Cc:</td>
<td>Martijn Jacobs</td>
</tr>
<tr>
<td>Owned by:</td>
<td>Martijn Jacobs</td>
</tr>
<tr>
<td>Version:</td>
<td>master</td>
</tr>
<tr>
<td>Keywords:</td>
<td>HFS+ OSX Testing</td>
</tr>
<tr>
<td>Triage Stage:</td>
<td>Ready for checkin</td>
</tr>
<tr>
<td>Needs documentation:</td>
<td>no</td>
</tr>
</tbody>
</table>

Opened 2 months ago  
Closed 2 months ago
#30323  closed Bug (fixed)

Django 2.2 autoreloader is failing intermittently (not using watchman)

<table>
<thead>
<tr>
<th>Reported by:</th>
<th>Mark Chackerian</th>
<th>Owned by:</th>
<th>Tom Forbes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component:</td>
<td>Utilities</td>
<td>Version:</td>
<td>2.2</td>
</tr>
<tr>
<td>Severity:</td>
<td>Release blocker</td>
<td>Keywords:</td>
<td>autoreloader</td>
</tr>
<tr>
<td>Cc:</td>
<td>Sammie S. Taunton, Keryn Knight, Tom Forbes</td>
<td>Triage Stage:</td>
<td>Ready for checkin</td>
</tr>
</tbody>
</table>
The aftermath

def watch_file():
    last_loop = time.time()
    while True:
        for path in get_files_to_watch():
            ...
            if previous_mtime is None and mtime > last_loop:
                exit(3)
            ...
        time.sleep(1)
    last_loop = time.time()
60% OF THE TIME

IT WORKS EVERY TIME
Conclusions:

Don't write your own autoloader.

Use this library:

https://github.com/Pylons/hupper
https://onfido.com/careers
Questions?

Tom Forbes - tom@tomforb.es