# 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()`?

```
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

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

```
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 = os.stat('/etc/password').st_mtime
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!**

```
p = pathlib.Path('test')
p.touch()
time.sleep(0.005)  # 5 milliseconds
p.touch()
```

# Filesystems can be *weird*.

**Network filesystems mess things up completely**

`os.stat()` **suddenly becomes expensive!**

# Watching files: A simple implementation

```
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

| Project | Test Count |
| --- | --- |
| Tornado | 2 |
| Flask | 3 |
| Pyramid | 6 |

# Making it testable

**Reloaders are infinite loops that run in threads and rely on a big ball of external state.**

# Generators!

# 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
```

# Generators!

```
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

```
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

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

**Not available in a virtualenv 😱**

# Making it efficient: Skipping the stdlib + third party packages

```
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 *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

```
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
```

# (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!

# The aftermath

## #30554 closed Uncategorized (invalid)

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

### Excessive logging by autoreload in v 2.2.1 and 2.2.2

| | | | |
|---|---|---|---|
| Reported by: | Phoebe Bright | Owned by: | nobody |
| Component: | Core (Other) | Version: | 2.2 |
| Severity: | Normal | Keywords: | |
| Cc: | Tom Forbes | Triage Stage: | Unreviewed |

## #30516 closed Bug (fixed)

Opened 4 weeks ago
Closed 4 weeks ago
Last modified 4 weeks ago

### Autoreloader crashes on re-raising exceptions with custom signature.

| | | | |
|---|---|---|---|
| Reported by: | Alan Trick | Owned by: | **Tom Forbes** |
| Component: | Utilities | Version: | 2.2 |
| Severity: | Release blocker | Keywords: | autoreload |
| Cc: | Tom Forbes | Triage Stage: | Accepted |

# The aftermath

## #30506 closed Bug (needsinfo)

Opened 4 weeks ago
Closed 4 weeks ago
Last modified 0 seconds ago

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

| | | | |
|---|---|---|---|
| Reported by: | Keryn Knight | Owned by: | nobody |
| Component: | Core (Management commands) | Version: | master |
| Severity: | Normal | Keywords: | |
| Cc: | Tom Forbes | Triage Stage: | Unreviewed |

## #30479 closed Bug (fixed)

Opened 6 weeks ago
Closed 4 weeks ago
Last modified 4 weeks ago

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

| | | | |
|---|---|---|---|
| Reported by: | Keryn Knight | Owned by: | **Tom Forbes** |
| Component: | Utilities | Version: | 2.2 |
| Severity: | Release blocker | Keywords: | autoreload |
| Cc: | Tom Forbes | Triage Stage: | Accepted |

# The aftermath

## #30372 closed Bug (needsinfo)

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

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

| | | | |
|---|---|---|---|
| Reported by: | Benjamin Schollnick | Owned by: | nobody |
| Component: | Core (Other) | Version: | 2.2 |
| Severity: | Normal | Keywords: | |
| Cc: | Tobias Kunze, Tom Forbes | Triage Stage: | Unreviewed |

## #30366 closed Bug (fixed)

Opened 2 months ago
Closed 2 months ago

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

| | | | |
|---|---|---|---|
| Reported by: | Martijn Jacobs | Owned by: | Martijn Jacobs |
| Component: | Testing framework | Version: | master |
| Severity: | Normal | Keywords: | HFS+ OSX Testing |
| Cc: | Martijn Jacobs | Triage Stage: | Ready for checkin |

# The aftermath

#30323 closed Bug (fixed)

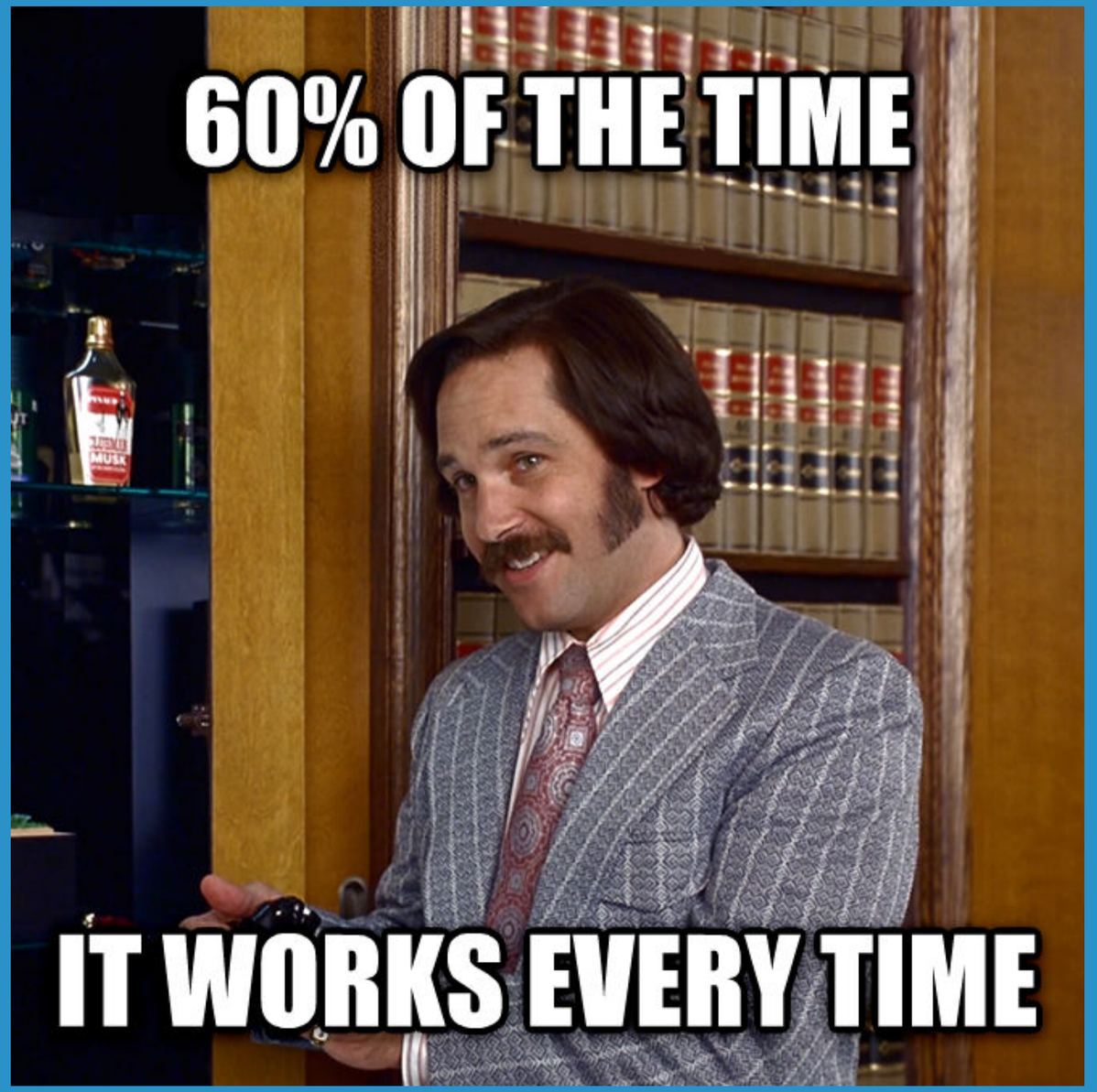
Opened 3 months ago
Closed 8 weeks ago
Last modified 8 weeks ago

## Django 2.2 autoreloader is failing intermittently (not using watchman)

| | | | |
|---|---|---|---|
| Reported by: | Mark Chackerian | Owned by: | **Tom Forbes** |
| Component: | Utilities | Version: | 2.2 |
| Severity: | Release blocker | Keywords: | autoreloader |
| Cc: | Sammie S. Taunton, Keryn Knight, Tom Forbes | Triage Stage: | Ready for checkin |

# 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**