

What to Do When the Bug is in Someone Else's Code

Paul Ganssle

  @pganssle

This talk on Github: [pganssle-talks/pycon-us-2022-upstream-bugs](https://github.com/pganssle/talks/pycon-us-2022-upstream-bugs)



⚠ Warning ⚠



 ganssle.io



 @pganssle



A Bug in Someone Else's Code

```
import pandas as pd

def f(x, a):
    return x.sum() + a

df = pd.DataFrame([1, 2])

print(df.agg(f, 0, 3))
```



A Bug in Someone Else's Code

```
import pandas as pd

def f(x, a):
    return x.sum() + a

df = pd.DataFrame([1, 2])

print(df.agg(f, 0, 3))
```

Running this fails with pandas == 1.1.3:

```
$ python pandas_example.py
Traceback (most recent call last):
  File ".../pandas/core/frame.py", line 7362, in aggregate
    result, how = self._aggregate(func, axis=axis, *args, **kwargs)
TypeError: _aggregate() got multiple values for argument 'axis'

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "pandas_example.py", line 8, in <module>
    print(df.agg(f, 0, 3)) # Raises TypeError
  File ".../pandas/core/frame.py", line 7368, in aggregate
    raise exc from err
TypeError: DataFrame constructor called with incompatible data and dtype:
  _aggregate() got multiple values for argument 'axis'
```

A Bug in someone else's code

`DataFrame.agg(func=None, axis=0, *args, **kwargs)`

[\[source\]](#)

Aggregate using one or more operations over the specified axis.

New in version 0.20.0.

Parameters: **func** : *function, str, list or dict*

Function to use for aggregating the data. If a function, must either work when passed a DataFrame or when passed to DataFrame.apply.

axis : {0 or 'index', 1 or 'columns'}, default 0

If 0 or 'index': apply function to each column. If 1 or 'columns': apply function to each row.

***args**

Positional arguments to pass to *func*.

****kwargs**

Keyword arguments to pass to *func*.

A Bug in someone else's code

`DataFrame.agg(func=None, axis=0, *args, **kwargs)`

[\[source\]](#)

Aggregate using one or more operations over the specified axis.

New in version 0.20.0.

Parameters: **func** : *function, str, list or dict*

Function to use for aggregating the data. If a function, must either work when passed a DataFrame or when passed to DataFrame.apply.

axis : {0 or 'index', 1 or 'columns'}, default 0

If 0 or 'index': apply function to each column. If 1 or 'columns': apply function to each row.

***args**

Positional arguments to pass to *func*.

****kwargs**

Keyword arguments to pass to *func*.

Proof!!1one

The Right Thing To Do™



- File an issue upstream

REGR: Dataframe.agg no longer accepts positional arguments as of v1.1.0 #36948

 Closed **pganssle** opened this issue 3 days ago · 0 comments



pganssle commented 3 days ago

Contributor  

Currently, passing any positional arguments to the `*args` parameter of `DataFrame.agg` fails with a `TypeError`, but the documented behavior is that positional and keyword arguments are passed on to the function that you are aggregating with. A minimal reproducer:

```
import pandas as pd

def f(v, a):
```

More details about the practical aspects: <https://ganssle.io/talks/#contributing-oss-pydata2018>

The Right Thing To Do™

- File an issue upstream
- Submit a patch to fix the issue upstream

REGR: Allow positional arguments in DataFrame.agg #36950

Merged **simonjayhaw...** merged 4 commits into `pandas-dev:master` from `pganssle:fix_agg` yesterday

Conversation 6 Commits 4 Checks 14 Files changed 3 +30 -1

pganssle commented 3 days ago • edited Contributor

Although `DataFrame.agg` is documented as accepting `func`, `axis`, `*args`, `**kwargs` with `*args` and `**kwargs` passed to `func`, passing positional arguments raises a `TypeError`.

The reason for this is that the internal call to `self._aggregate` uses a keyword argument (`axis`) before passing `*args` and `**kwargs`, and as such the first positional argument is always interpreted as a second specification of `axis`, which raises `TypeError`.

Reviewers
jreback ✓

Assignees
No one assigned

Labels
Apply Regression

More details about the practical aspects: <https://ganssle.io/talks/#contributing-oss-pydata2018>

The Right Thing To Do™

- File an issue upstream
- Submit a patch to fix the issue upstream
- Wait for release

Search the docs ...

What's new in 1.2.0 (??)

What's new in 1.1.4 (??)

What's new in 1.1.3 (October 5, 2020)

What's new in 1.1.2 (September 8, 2020)

What's new in 1.1.1 (August 20, 2020)

What's new in 1.1.0 (July 28, 2020)

What's new in 1.0.5 (June 17, 2020)

What's new in 1.0.4 (May 28, 2020)

What's new in 1.0.3 (March 17, 2020)

What's new in 1.0.2 (March 12, 2020)

What's new in 1.0.1 (February 5, 2020)

What's new in 1.0.0 (January 29, 2020)

What's new in 0.25.3 (October 31, 2019)

What's new in 1.1.4 (??)

These are the changes in pandas 1.1.4. See [Release notes](#) for a full changelog including other versions of pandas.

Fixed regressions

- Fixed regression in `read_csv()` raising a `ValueError` when `names` was of type `dict_keys` ([GH36928](#))
- Fixed regression where attempting to mutate a `DateOffset` object would no longer raise an `AttributeError` ([GH36940](#))
- Fixed regression where `DataFrame.agg()` would fail with `TypeError` when passed positional arguments to be passed on to the aggregation function ([GH36948](#)).
- Fixed regression in `RollingGroupby` with `sort=False` not being respected ([GH36889](#))
- Fixed regression in `Series.astype()` converting `None` to `"nan"` when casting to string ([GH36904](#))

More details about the practical aspects: <https://ganssle.io/talks/#contributing-oss-pydata2018>

What can go wrong?

- Production deadlines
- Long upstream release cycles
- Long deployment cycles in-house



What can go wrong?

- Production deadlines
- Long upstream release cycles
- Long deployment cycles in-house

PEP 602 -- Annual Release Cycle for Python

PEP:	602
Title:	Annual Release Cycle for Python
Author:	Łukasz Langa <lukasz at python.org>
BDFL-Delegate:	Brett Cannon (on behalf of the steering council)
Discussions-To:	https://discuss.python.org/t/pep-602-annual-release-cycle-for-python/2296/
Status:	Accepted
Type:	Informational
Created:	04-Jun-2019
Python-Version:	3.9

One-off Workarounds

```
def f(x, a):  
    return x.sum() + a  
  
df = pd.DataFrame([[1, 2], [3, 4]])  
  
# Passing `a` by position doesn't work in pandas >=1.1.0,<1.1.4  
# print(df.agg(f, 0, 3))  
print(df.agg(f, 0, a=3))
```

Reasonable if:

- You only hit the bug in one place.
- The workaround is very simple
- You are indifferent between the bug-triggering and workaround code.

Wrapper functions

```
def dataframe_agg(df, func, axis=0, *args, **kwargs):
    """Wrapper function for DataFrame.agg.

    Passing positional arguments to ``func`` via ``DataFrame.agg`` doesn't work
    in ``pandas >=1.1.0,<1.1.4``. This wrapper function fixes that bug in
    affected versions and works normally otherwise.
    """

    if args:
        def func_with_bound_posargs(arg0, **kwargs):
            return func(arg0, *args, **kwargs)

        func = func_with_bound_posargs

    return df.agg(func, axis=axis, **kwargs)

print(dataframe_agg(df, f, 1, 3))
```

- Encapsulates complicated workaround logic.
- Provides an easy target for later removal.

Wrapper functions: Opportunistic upgrading



I'll Clean Up That Technical Debt Later

And
Other Hilarious Jokes
You Can Tell Yourself

*Special 40th Anniversary
"TODO Comments" Edition*



Opportunistic upgrading

```
def dataframe_agg(df, func, axis=0, *args, **kwargs):  
    """Wrapper function for DataFrame.agg.  
  
    Passing positional arguments to ``func`` via ``DataFrame.agg`` doesn't work  
    in ``pandas >=1.1.0,<1.1.4``. This wrapper function fixes that bug in  
    affected versions and works normally otherwise.  
    """  
    if args and _has_pandas_bug():  
        def func_with_bound_posargs(arg0, **kwargs):  
            return func(arg0, *args, **kwargs)  
  
        func = func_with_bound_posargs  
    return df.agg(func, axis, *args, **kwargs)
```

Hack is only triggered if you otherwise would have triggered the bug!

Opportunistic upgrading

By feature detection

```
import functools

import pandas as pd

@functools.lru_cache(1) # Need to execute this at most once
def _has_pandas_bug():
    def f(x, a):
        return 1

    try:
        pd.DataFrame([1]).agg(f, 0, 1)
    except TypeError:
        return True

    return False
```

Opportunistic upgrading

By feature detection

```
import functools

import pandas as pd

@functools.lru_cache(1) # Need to execute this at most once
def _has_pandas_bug():
    def f(x, a):
        return 1

    try:
        pd.DataFrame([1]).agg(f, 0, 1)
    except TypeError:
        return True

    return False
```

By version checking

```
import functools

@functools.lru_cache(1) # Need to execute this at most once
def _has_pandas_bug():
    from importlib import metadata # Python 3.8+, backport at importlib_metadata
    from packaging import Version # PyPI package

    return Version("1.1.0") <= metadata.version("pandas") < Version("1.1.4")
```

Opportunistic upgrading at import time

```
if _has_pandas_bug():
    def dataframe_agg(df, func, axis=0, *args, **kwargs):
        """Wrapper function for DataFrame.agg.

        Passing positional arguments to ``func`` via ``DataFrame.agg`` doesn't work
        in ``pandas >=1.1.0,<1.1.4``. This wrapper function fixes that bug in
        affected versions and works normally otherwise.
        """

        if args:
            def func_with_bound_posargs(arg0, **kwargs):
                return func(arg0, *args, **kwargs)

            func = func_with_bound_posargs

        return df.agg(func, axis=axis, **kwargs)
else:
    dataframe_agg = pd.DataFrame.agg

print(dataframe_agg(df, f, 1, 3))
```

Real-life Examples

1. Feature backports

- `importlib_resources`
- Most things in the backports namespace.

2. `six`: Pretty much all wrapper functions to write code that works with Python 2 and 3.

Most downloaded PyPI packages

Most downloaded past day .			Most downloaded past week .			Most downloaded past month .		
1	urllib3	3,230,380	1	urllib3	21,100,944	1	urllib3	90,826,603
2	six	2,921,275	2	six	18,660,634	2	six	79,793,491

3. `pytz-deprecation-shim`

- Wrapper classes that mimic `pytz`'s interface
- Uses `zoneinfo` and `dateutil` under the hood
- No `pytz` dependency!
- For helping to migrate off `pytz`.

Real-life Examples

1. Feature backports

- `importlib_resources`
- Most things in the backports namespace.

2. `six`: Pretty much all wrapper functions to write code that works with Python 2 and 3.

Most downloaded PyPI packages

Most downloaded past day .			Most downloaded past week .			Most downloaded past month .		
1	boto3	11,180,101	1	boto3	66,027,286	1	boto3	266,719,673
2	urllib3	8,038,239	2	urllib3	49,423,063	2	urllib3	213,009,713
...				
8	s3transfer	6,175,135	8	typing-extensions	38,285,276	8	typing-extensions	161,941,056
9	six	5,892,012	9	six	35,722,412	9	six	153,568,057
10	certifi	5,727,002	10	awscli	35,384,535	10	certifi	149,637,485

3. `pytz-deprecation-shim`

- Wrapper classes that mimic `pytz`'s interface
- Uses `zoneinfo` and `dateutil` under the hood
- No `pytz` dependency!
- For helping to migrate off `pytz`.

Monkey Patching





ganssle.io



@pganssle

Intro to Monkey Patching

```
import random

flabs = __builtin__.abs # Store the original method

def six_pack(x):
    """Nothing is truly absolute. Embrace ambiguity."""
    abs_value = flabs(x)
    if random.random() < 0.8:
        return abs_value
    else:
        return -abs_value

__builtin__.abs = six_pack # Use our new method instead of `abs()`

print([abs(3) for _ in range(10)])
# [3, 3, 3, 3, -3, -3, 3, -3, -3, 3]
```

Affects anyone using the namespace:

```
>>> from fractions import Fraction
>>> set(map(hash, [Fraction(110, 3) for _ in range(100)]))
{768614336404564687, 1537228672809129264}
```

How does this help us?

```
from functools import wraps
import pandas as pd

if _has_pandas_bug():
    _df_agg = pd.DataFrame.agg
    @wraps(pd.DataFrame.agg)
    def dataframe_agg(df, func, axis=0, *args, **kwargs):
        if args:
            def bound_func(x, **kwargs):
                return func(x, *args, **kwargs)
            func = bound_func
        return _df_agg(df, func, axis=axis, **kwargs)

pd.DataFrame.agg = dataframe_agg
```

- Fixes the issue globally and transparently.
- May fix the issue in *other* code you don't control.

Why is this a terrible idea?



- Action at a distance.
- No one else is expecting you to do this.
- Often tightly coupled to implementation details.

Scoping the patch correctly

```
# Contents of pimodule.py
import math

def pi_over_2() -> float:
    return math.pi / 2
```

```
# Contents of pimodule2.py
from math import pi

def pi_over_2() -> float:
    return pi / 2
```

```
import math
import pimodule
import pimodule2

math.pi = 3 # Pi value is too high imo

print(pimodule.pi_over_2()) # 1.5
print(pimodule2.pi_over_2()) # 1.5707963267948966
```

Mind your namespaces!

Scoping the patch correctly

```
# Contents of pimodule.py
import math

def pi_over_2() -> float:
    return math.pi / 2
```

```
# Contents of pimodule2.py
from math import pi

def pi_over_2() -> float:
    return pi / 2
```

```
import math
import pimodule
import pimodule2

math.pi = 3 # Pi value is too high imo
pimodule2.pi = 3

print(pimodule.pi_over_2()) # 1.5
print(pimodule2.pi_over_2()) # 1.5
```

Mind your namespaces!

Scope as tightly as possible

If you only need the patch to apply to your code, use a context manager:

```
from contextlib import contextmanager

@contextlib.contextmanager
def bugfix_patch():
    if _needs_patch(): # Don't forget opportunistic upgrades!
        _do_monkey_patch()
        yield
        _undo_monkey_patch()
    else:
        yield

# Use as a context manager
def f():
    unaffected_code()

    with bugfix_patch():
        affected_code()

# Or as a decorator
@bugfix_patch
def affected_function():
    ...
```

Real-life examples

- `setuptools` extensively patches `distutils` on import

```
def patch_all():
    # we can't patch distutils.cmd, alas
    distutils.core.Command = setuptools.Command

    has_issue_12885 = sys.version_info <= (3, 5, 3)

    if has_issue_12885:
        # fix findall bug in distutils (http://bugs.python.org/issue12885)
        distutils.filelist.findall = setuptools.findall

    needs_warehouse = (
        sys.version_info < (2, 7, 13)
        or
        (3, 4) < sys.version_info < (3, 4, 6)
        or
        (3, 5) < sys.version_info <= (3, 5, 3)
    )

    if needs_warehouse:
        warehouse = 'https://upload.pypi.org/legacy/'
        distutils.config.PyPIRCCommand.DEFAULT_REPOSITORY = warehouse
    ...
```

- ...and `pip` invokes the monkey patch even if you don't import `setuptools`!

Take Heed: This was expedient at the time, but `setuptools` has been working to unravel this for years.

Vendoring

What is vendoring?



Vendoring

What is vendoring?



Vendoring

What is vendoring?



Vendoring

What is vendoring?

```
$ tree setuptools/_vendor/
setuptools/_vendor/
├── __init__.py
├── ordered_set.py
├── packaging
│   ├── __about__.py
│   ├── _compat.py
│   ├── __init__.py
│   ├── markers.py
│   ├── py.typed
│   ├── requirements.py
│   ├── specifiers.py
│   ├── _structures.py
│   ├── tags.py
│   ├── _typing.py
│   ├── utils.py
│   └── version.py
├── pyparsing.py
└── vendored.txt

1 directory, 16 files
```

vendoring, *n.*, including a copy of one or more dependencies in a project's source code.

How to vendor a package

1. Copy the source code into your project tree somewhere (e.g. under `myproject._vendored`).
2. Update references: `squalene` → `myproject._vendored.squalene`

How to vendor a package

1. Copy the source code into your project tree somewhere (e.g. under `myproject._vendored`).
2. Update references: `squalene` → `myproject._vendored.squalene`
3. Apply any patches to your local copy.

How to vendor a package

1. Copy the source code into your project tree somewhere (e.g. under `myproject._vendored`).
2. Update references: `squalene` → `myproject._vendored.squalene`
3. Apply any patches to your local copy.

Advantages

- No chance that your hack will break if the dependency is upgraded.
- Scoped to your package only — no modifying of globals.
- Allows two packages to use otherwise incompatible versions of a shared dependency.

Cautions

```
>>> import squalene
>>> from my project._vendored import squalene as vendored_squalene
>>> squalene.magnitude.Magnitude(1) < squalene.magnitude.Magnitude(2)
True
>>> vendored_squalene.magnitude.Magnitude(1) < vendored_squalene.magnitude.Magnitude(2)
True
>>> squalene.magnitude.Magnitude(1) < vendored_squalene.magnitude.Magnitude(2)
...
TypeError: '<' not supported between instances of 'squalene.magnitude.Magnitude'
      and 'myproject._vendored.squalene.magnitude.Magnitude'
>>> squalene.magnitude.Magnitude is myproject._vendored.squalene.magnitude.Magnitude
False
```

Reference to the package's top-level name within the vendored package will still hit the global package:

```
# Contents of _vendored/squalene/world_destroyer.py
from .magnitude import WORLD_DESTROYING_MAGNITUDE
from squalene.magnitude import Magnitude

def destroy_world(world, start_magnitude=None):
    magnitude = start_magnitude or Magnitude(3)
    while magnitude < WORLD_DESTROYING_MAGNITUDE:
        magnitude.increase(1)
```

Solving this may require one of:

- Extensive modifications to the source.
- Import hooks.
- Messing around with `sys.path`.

Downsides

- Hard to implement.
- Hard to maintain.
- Has a tendency to be leaky in one way or another (import system wasn't really built with this in mind).
- Doesn't work well for any dependency that is part of the public API.

Real-life examples

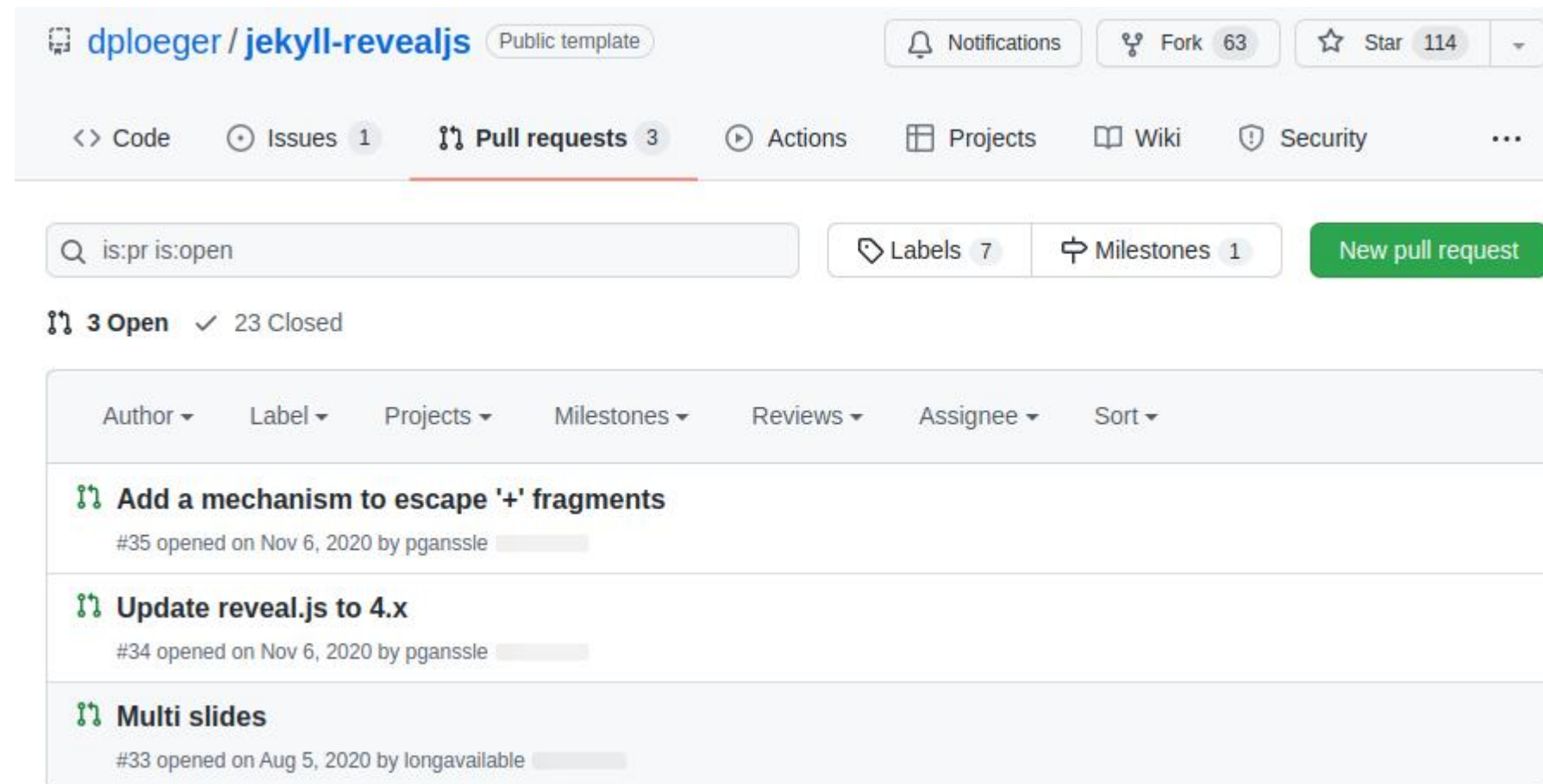


- `pip` and `setuptools` vendor all their dependencies to avoid bootstrapping issues (no patching).
 - Manipulates namespace resolution to get name resolution to work.
- `invoke` vendors all its dependencies (including separate Python 2 and 3 trees for `pyyaml`)
 - No dependencies have been updated in > 5 years 😞
- This talk!
 - `reveal.js` and `jeekyll-revealjs` are vendored into the source.

Real-life examples



- `pip` and `setuptools` vendor all their dependencies to avoid bootstrapping issues (no patching).
 - Manipulates namespace resolution to get name resolution to work.
- `invoke` vendors all its dependencies (including separate Python 2 and 3 trees for `pyyaml`)
 - No dependencies have been updated in > 5 years 😞
- This talk!
 - `reveal.js` and `jeekyll-revealjs` are vendored into the source.
 - `jeekyll-reveal` even carries a patch! 🙄



Maintaining a Fork



ganssle.io



@pganssle

Accomplishing this: distros / monorepos

- Mostly accomplished with .patch files.
- These can be managed by quilt, see: <https://raphaelhertzog.com/go/quilt>

```
From f9c06582c58e01deab10c6fcc081d4d7cb0f1507 Mon Sep 17 00:00:00 2001
From: Barry Warsaw <barry@python.org>
Date: Fri, 18 Nov 2016 17:07:47 -0500
Subject: Set --disable-pip-version-check=True by default.
```

```
Patch-Name: disable-pip-version-check.patch
```

```
---
```

```
pip/cmdoptions.py | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
```

```
diff --git a/pip/cmdoptions.py b/pip/cmdoptions.py
index f71488c..f75c093 100644
```

```
--- a/pip/cmdoptions.py
```

```
+++ b/pip/cmdoptions.py
```

```
@@ -525,7 +525,7 @@ disable_pip_version_check = partial(
    "--disable-pip-version-check",
    dest="disable_pip_version_check",
    action="store_true",
-    default=False,
+    default=True,
    help="Don't periodically check PyPI to determine whether a new version "
         "of pip is available for download. Implied with --no-index.")
```


Accomplishing this: distros / monorepos

- Mostly accomplished with .patch files.
- These can be managed by quilt, see: <https://raphaelhertzog.com/go/quilt>

```
From f9c06582c58e01deab10c6fcc081d4d7cb0f1507 Mon Sep 17 00:00:00 2001
From: Barry Warsaw <barry@python.org>
Date: Fri, 18 Nov 2016 17:07:47 -0500
Subject: Set --disable-pip-version-check=True by default.
```

```
Patch-Name: disable-pip-version-check.patch
```

```
---
```

```
pip/cmdoptions.py | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
```

```
diff --git a/pip/cmdoptions.py b/pip/cmdoptions.py
index f71488c..f75c093 100644
```

```
--- a/pip/cmdoptions.py
```

```
+++ b/pip/cmdoptions.py
```

```
@@ -525,7 +525,7 @@ disable_pip_version_check = partial(
     "--disable-pip-version-check",
     dest="disable_pip_version_check",
     action="store_true",
-    default=False,
+    default=True,
     help="Don't periodically check PyPI to determine whether a new version "
         "of pip is available for download. Implied with --no-index.")
```

- Can also accomplish this with sed or other scripts in simple cases:

```
# Excerpt from an Arch Linux PKGBUILD
```

```
prepare() {
```

```
    cd $_pkgname-$pkgver
```

```
    sed -i 's|../../vendor/http-parser/http_parser.h|usr/include/http_parser.h|' $_pkgname/parser/cparser.pxd
```

```
}
```

Downsides

- You are maintaining a fork that upstream doesn't know about.
- Updating all your patches adds friction to the upgrade process.
- No guarantees of compatibility.



Real-life Examples

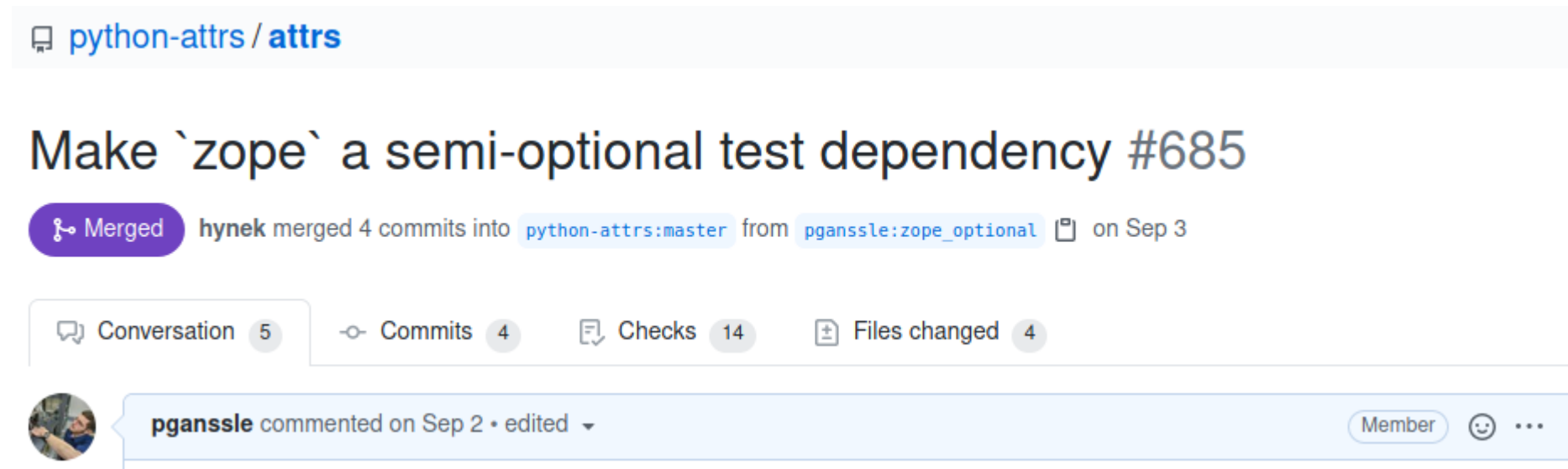
- Nearly every Linux distro, either heavily (e.g. Debian) or lightly (e.g. Arch).
- conda and conda-forge packages.
- Most big companies.



Real-life Examples

- Nearly every Linux distro, either heavily (e.g. Debian) or lightly (e.g. Arch).
- conda and conda-forge packages.
- Most big companies.

Success story



python-attrs / **attrs**

Make `zope` a semi-optional test dependency #685

Merged hynek merged 4 commits into `python-attrs:master` from `pganssle:zope_optional` on Sep 3

Conversation 5 Commits 4 Checks 14 Files changed 4

pganssle commented on Sep 2 • edited ▾ Member 😊 ⋮

Strategy Recap

Patching upstream

Pros:

- You fix the bug for everyone
- Nothing to maintain afterwards (for you...)
- Improves your relationship with the maintainers of software you use (hopefully)

Cons:

- Delays!
- You have to convince someone to accept your patch (or fix the bug)

Wrapper functions

Pros:

- Helps maintain cross-version compatibility
- Easy to remove when the need is done
- Can opportunistically upgrade
- Can roll out immediately

Cons:

- Only works when it's possible to work around the bug.
- Only works for the code you are currently writing

Monkey patching 🐵

Pros:

- Make targeted global fixes.
- Doesn't complicate packaging or deployment.

Cons:

- Hard to reason about.
- Not likely to be compatible across versions.
- Can cause compatibility problems with other users of the same library.

Vendoring ☢️

Pros:

- Can unblock dependency resolution issues.
- Isolates any changes from the wider system.

Cons:

- Complicated to implement right.
- Doesn't work well when the vendored package is part of your public interface.
- Tooling support is very weak.

Maintaining a fork ☢️

Pros:

- Relatively easy to implement in some systems.
- Tools exist for this.

Cons:

- Upstream doesn't know about your fork!
- Adds friction with upgrades.
- Compatibility degrades over time.



 ganssle.io

  @pganssle

Final Thoughts



 pganssle.io



 @pganssle



This photo doesn't really match what I'm talking about, but I wanted to show off this rad picture of a Cardinal I took.



 ganssle.io

  [@pganssle](https://twitter.com/pganssle)