django-nose Documentation

Release 1.4.6

Jeff Balogh and the django-nose team

Oct 03, 2018

Contents

1 Installation **3**

2 Development **5**

3 Contents **7**

3.1 Installation 7
3.2 Usage 7
3.3 Upgrading Django 10
3.4 Contributing 11
3.5 Credits 13
3.6 Changelog 15

4 Indices and tables **19**

django-nose provides all the goodness of nose in your Django tests, like:

- Testing just your apps by default, not all the standard ones that happen to be in `INSTALLED_APPS`
- Running the tests in one or more specific modules (or apps, or classes, or folders, or just running a specific test)
- Obviating the need to import all your tests into `tests/__init__.py`. This not only saves busy-work but also eliminates the possibility of accidentally shadowing test classes.
- Taking advantage of all the useful nose plugins

It also provides:

- Fixture bundling, an optional feature which speeds up your fixture-based tests by a factor of 4
- Reuse of previously created test DBs, cutting 10 seconds off startup time
- Hygienic TransactionTestCases, which can save you a DB flush per test
- Support for various databases. Tested with MySQL, PostgreSQL, and SQLite. Others should work as well.

django-nose requires nose 1.2.1 or later, and the latest release is recommended. It follows the Django's support policy, supporting:

- Django 1.8 (LTS) with Python 2.7, 3.4, or 3.5
- Django 1.9 with Python 2.7, 3.4, or 3.5
- Django 1.10 with Python 2.7, 3.4, or 3.5
- Django 1.11 (LTS) with Python 2.7, 3.4, 3.5, or 3.6
- Django 2.0 with Python 3.4, 3.5, 3.6, or 3.7
- Django 2.1 with Python 3.5, 3.6, or 3.7

CHAPTER 1

Installation

You can get django-nose from PyPI with... :

```
$ pip install django-nose
```

The development version can be installed with... :

```
$ pip install -e git://github.com/django-nose/django-nose.git#egg=django-nose
```

Since django-nose extends Django's built-in test command, you should add it to your `INSTALLED_APPS` in `settings.py`:

```
INSTALLED_APPS = (
    ...
    'django_nose',
    ...
)
```

Then set `TEST_RUNNER` in `settings.py`:

```
TEST_RUNNER = 'django_nose.NoseTestSuiteRunner'
```

CHAPTER 2

Development

Code https://github.com/django-nose/django-nose

Issues https://github.com/django-nose/django-nose/issues?state=open

Docs https://django-nose.readthedocs.io

CHAPTER 3

Contents

3.1 Installation

You can get django-nose from PyPI with... :

```
$ pip install django-nose
```

The development version can be installed with... :

```
$ pip install -e git://github.com/django-nose/django-nose.git#egg=django-nose
```

Since django-nose extends Django's built-in test command, you should add it to your `INSTALLED_APPS` in `settings.py`:

```
INSTALLED_APPS = (
    ...
    'django_nose',
    ...
)
```

Then set `TEST_RUNNER` in `settings.py`:

```
TEST_RUNNER = 'django_nose.NoseTestSuiteRunner'
```

3.2 Usage

The day-to-day use of django-nose is mostly transparent; just run `./manage.py test` as usual.

See `./manage.py help test` for all the options nose provides, and look to the nose docs for more help with nose.

3.2.1 Enabling Database Reuse

Warning: There are several open issues with this feature, including reports of data loss.

You can save several seconds at the beginning and end of your test suite by reusing the test database from the last run. To do this, set the environment variable `REUSE_DB` to 1:

```
REUSE_DB=1 ./manage.py test
```

The one new wrinkle is that, whenever your DB schema changes, you should leave the flag off the next time you run tests. This will cue the test runner to reinitialize the test database.

Also, REUSE_DB is not compatible with TransactionTestCases that leave junk in the DB, so be sure to make your TransactionTestCases hygienic (see below) if you want to use it.

3.2.2 Enabling Fast Fixtures

Warning: There are several known issues with this feature.

django-nose includes a fixture bundler which drastically speeds up your tests by eliminating redundant setup of Django test fixtures. To use it...

1. Subclass `django_nose.FastFixtureTestCase` instead of `django.test.TestCase`. (I like to import it `as TestCase` in my project's `tests/__init__.py` and then import it from there into my actual tests. Then it's easy to sub the base class in and out.) This alone will cause fixtures to load once per class rather than once per test.
2. Activate fixture bundling by passing the `--with-fixture-bundling` option to `./manage.py test`. This loads each unique set of fixtures only once, even across class, module, and app boundaries.

How Fixture Bundling Works

The fixture bundler reorders your test classes so that ones with identical sets of fixtures run adjacently. It then advises the first of each series to load the fixtures once for all of them (and the remaining ones not to bother). It also advises the last to tear them down. Depending on the size and repetition of your fixtures, you can expect a 25% to 50% speed increase.

Incidentally, the author prefers to avoid Django fixtures, as they encourage irrelevant coupling between tests and make tests harder to comprehend and modify. For future tests, it is better to use the "model maker" pattern, creating DB objects programmatically. This way, tests avoid setup they don't need, and there is a clearer tie between a test and the exact state it requires. The fixture bundler is intended to make existing tests, which have already committed to fixtures, more tolerable.

Troubleshooting

If using `--with-fixture-bundling` causes test failures, it likely indicates an order dependency between some of your tests. Here are the most frequent sources of state leakage we have encountered:

- Locale activation, which is maintained in a threadlocal variable. Be sure to reset your locale selection between tests.

- memcached contents. Be sure to flush between tests. Many test superclasses do this automatically.

It's also possible that you have `post_save` signal handlers which create additional database rows while loading the fixtures. `FastFixtureTestCase` isn't yet smart enough to notice this and clean up after it, so you'll have to go back to plain old `TestCase` for now.

Exempting A Class From Bundling

In some unusual cases, it is desirable to exempt a test class from fixture bundling, forcing it to set up and tear down its fixtures at the class boundaries. For example, we might have a `TestCase` subclass which sets up some state outside the DB in `setUpClass` and tears it down in `tearDownClass`, and it might not be possible to adapt those routines to heed the advice of the fixture bundler. In such a case, simply set the `exempt_from_fixture_bundling` attribute of the test class to `True`.

3.2.3 Speedy Hygienic TransactionTestCases

Unlike the stock Django test runner, django-nose lets you write custom TransactionTestCase subclasses which expect to start with an unmarred DB, saving an entire DB flush per test.

Background

The default Django TransactionTestCase class can leave the DB in an unclean state when it's done. To compensate, TransactionTestCase does a time-consuming flush of the DB *before* each test to ensure it begins with a clean slate. Django's stock test runner then runs TransactionTestCases last so they don't wreck the environment for better-behaved tests. django-nose replicates this behavior.

Escaping the Grime

Some people, however, have made subclasses of TransactionTestCase that clean up after themselves (and can do so efficiently, since they know what they've changed). Like TestCase, these may assume they start with a clean DB. However, any TransactionTestCases that run before them and leave a mess could cause them to fail spuriously.

django-nose offers to fix this. If you include a special attribute on your well-behaved TransactionTestCase…

```
class MyNiceTestCase(TransactionTestCase):
    cleans_up_after_itself = True
```

…django-nose will run it before any of those nasty, trash-spewing test cases. You can thus enjoy a big speed boost any time you make a TransactionTestCase clean up after itself: skipping a whole DB flush before every test. With a large schema, this can save minutes of IO.

django-nose's own FastFixtureTestCase uses this feature, even though it ultimately acts more like a TestCase than a TransactionTestCase.

3.2.4 Test-Only Models

If you have a model that is used only by tests (for example, to test an abstract model base class), you can put it in any file that's imported in the course of loading tests. For example, if the tests that need it are in `test_models.py`, you can put the model in there, too. django-nose will make sure its DB table gets created.

3.2.5 Assertions

`django-nose.tools` provides pep8 versions of Django's TestCase asserts and some of its own as functions.

```
assert_redirects(response, expected_url, status_code=302, target_status_code=200,␣
↪host=None, msg_prefix='')

assert_contains(response, text, count=None, status_code=200, msg_prefix='')
assert_not_contains(response, text, count=None, status_code=200, msg_prefix='')

assert_form_error(response, form, field, errors, msg_prefix='')

assert_template_used(response, template_name, msg_prefix='')
assert_template_not_used(response, template_name, msg_prefix='')

assert_queryset_equal(qs, values, transform=repr)

assert_num_queries(num, func=None, *args, **kwargs)

assert_code(response, status_code, msg_prefix='')

assert_ok(response, msg_prefix='')

assert_mail_count(count, msg=None)
```

3.2.6 Always Passing The Same Options

To always set the same command line options you can use a nose.cfg or setup.cfg (as usual) or you can specify them in settings.py like this:

```
NOSE_ARGS = ['--failed', '--stop']
```

3.2.7 Custom Plugins

If you need to make custom plugins, you can define each plugin class somewhere within your app and load them from settings.py like this:

```
NOSE_PLUGINS = [
    'yourapp.tests.plugins.SystematicDysfunctioner',
    # ...
]
```

Just like middleware or anything else, each string must be a dot-separated, importable path to an actual class. Each plugin class will be instantiated and added to the Nose test runner.

3.3 Upgrading Django

3.3.1 Upgrading from Django <= 1.3 to Django 1.4

In versions of Django < 1.4 the project folder was in fact a python package as well (note the __init__.py in your project root). In Django 1.4, there is no such file and thus the project is not a python module.

When you upgrade your Django project to the Django 1.4 layout, you need to remove the __init__.py file in the root of your project (and move any python files that reside there other than the manage.py) otherwise you will get a 'ImportError: No module named urls' exception.

This happens because Nose will intelligently try to populate your sys.path, and in this particular case includes your parent directory if your project has a __init__.py file (see: https://github.com/nose-devs/nose/blob/release_1.1.2/nose/importer.py#L134).

This means that even though you have set up your directory structure properly and set your *ROOT_URLCONF='my_project.urls'* to match the new structure, when running django-nose's test runner it will try to find your urls.py file in *'my_project.my_project.urls'*.

3.3.2 Upgrading from Django < 1.2

Django 1.2 switches to a class-based test runner. To use django-nose with Django 1.2, change your `TEST_RUNNER` from `django_nose.run_tests` to `django_nose.NoseTestSuiteRunner`.

`django_nose.run_tests` will continue to work in Django 1.2 but will raise a warning. In Django 1.3, it will stop working.

If you were using `django_nose.run_gis_tests`, you should also switch to `django_nose.NoseTestSuiteRunner` and use one of the spatial backends in your `DATABASES` settings.

3.3.3 Django 1.1

If you want to use django-nose with Django 1.1, use https://github.com/django-nose/django-nose/tree/django-1.1 or http://pypi.python.org/pypi/django-nose/0.0.3.

3.3.4 Django 1.0

django-nose does not support Django 1.0.

3.4 Contributing

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

You can contribute in many ways:

3.4.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/django-nose/django-nose/issues.

If you are reporting a bug, please include:

- The version of django, nose, and django-nose you are using, and any other applicable packages (`pip freeze` will show current versions)
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

When someone submits a pull request to fix your bug, please try it out and report if it worked for you.

Fix Bugs

Look through the GitHub issues for bugs. Anything untagged or tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything untagged ot tagged with "feature" is open to whoever wants to implement it.

django-nose is built on nose, which supports plugins. Consider implementing your feature as a plugin, maintained by the community using that feature, rather than adding to the django-nose codebase.

Write Documentation

django-nose could always use more documentation, whether as part of the official django-nose, as code comments, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/django-nose/django-nose/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 :)

3.4.2 Get Started!

Ready to contribute? Here's how to set up django-nose for local development.

1. Fork the *django-nose* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-nose.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-nose
$ cd django-nose/
$ pip install -r requirements.txt
$ ./manage.py migrate
```

4. Create a branch for local development:

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

   Now you can make your changes locally.

5. Make sure existing tests continue to pass with your new code:

```
$ make qa
```

6. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ make qa-all
```

6. 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
```

7. Submit a pull request through the GitHub website.

3.4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should be in a branch.
2. The pull request should include tests.
3. You agree to license your contribution under the BSD license.
4. If the pull request adds functionality, the docs should be updated.
5. Make liberal use of *git rebase* to ensure clean commits on top of master.
6. The pull request should pass QA tests and work for supported Python / Django combinations. Check https://travis-ci.org/django-nose/django-nose/pull_requests and make sure that the tests pass for all supported Python versions.

3.4.4 Tips

The django-nose testapp uses django-nose, so all of the features are available. To run a subset of tests:

```
$ ./manage.py test testapp/tests.py
```

To mark failed tests:

```
$ ./manage.py test --failed
```

To re-run only the failed tests:

```
$ ./manage.py test --failed
```

3.5 Credits

django-nose was created by Jeff Balogh in 2009, which is a really long time ago in the Django world. It keeps running because of the contributions of volunteers. Thank you to everyone who uses django-nose, keeps your projects up-to-date, files detailed bugs, and submits patches.

3.5.1 Maintainers

- Jeff Balogh (jbalogh)
- Erik Rose (erikrose)
- James Socol (jscol)
- Rob Hudson (robhudson)
- John Whitlock (jwhitlock)

3.5.2 Contributors

These non-maintainers have contributed code to a django-nose release:

- Adam DePue (adepue)
- Alexey Evseev (st4lk)
- Alex (alexjg)
- Antti Kaihola (akaihola)
- Ash Christopher (ashchristopher)
- Blake Winton (bwinton)
- Brad Pitcher (brad)
- Camilo Nova (camilonova)
- Carl Meyer (carljm)
- Conrado Buhrer (conrado)
- David Baumgold (singingwolfboy)
- David Cramer (dcramer)
- Dmitry Gladkov (dgladkov)
- Ederson Mota (edrmp)
- Eric Zarowny (ezarowny)
- Eron Villarreal (eroninjapan)
- Fred Wenzel (fwenzel)
- Fábio Santos (fabiosantoscode)
- Ionel Cristian Mărieș (ionelmc)
- Jeremy Satterfield (jsatt)
- Johan Euphrosine (proppy)
- Kyle Robertson (dvelyk)
- Marius Gedminas (mgedmin)
- Martin Chase (outofculture)
- Matthias Bauer (moeffju)
- Michael Elsdörfer (miracle2k)

- Michael Kelly (Osmose)
- Peter Baumgartner (ipmb)
- Radek Simko (radeksimko)
- Ramiro Morales (ramiro)
- Rob Madole (robmadole)
- Roger Hu (rogerhu)
- Ross Lawley (rozza)
- Scott Sexton (scottsexton)
- Stephen Burrows (melinath)
- Sverre Johansen (sverrejoh)
- Tim Child (timc3)
- Walter Doekes (wdoekes)
- Will Kahn-Greene (willkg)
- Yegor Roganov (roganov)

3.6 Changelog

3.6.1 1.4.6 (2018-10-03)

- Document Django 2.0 and 2.1 support, no changes needed
- Document Python 3.7 support

3.6.2 1.4.5 (2017-08-22)

- Add Django 1.11 support

3.6.3 1.4.4 (2016-06-27)

- Add Django 1.10 support
- Drop Django 1.4 - 1.7, and Python 2.6 support
- Drop South support

3.6.4 1.4.3 (2015-12-28)

- Add Django 1.9 support
- Support long options without equals signs, such as "–attr selected"
- Support nose plugins using callback options
- Support nose options without default values (jsatt)
- Remove Django from install dependencies, to avoid accidental upgrades (jsocol, willkg)

- Setting REUSE_DB to an empty value now disables REUSE_DB, instead of enabling it (wdoekes)

3.6.5 1.4.2 (2015-10-07)

- Warn against using REUSE_DB=1 and FastFixtureTestCase in docs
- REUSE_DB=1 uses new transaction management in Django 1.7, 1.8 (scottsexton)
- Try to avoid accidentally using production database with REUSE_DB=1 (alexjg, eroninjapan)
- Supported Django versions limited to current supported Django version 1.4, 1.7, and 1.8, as well as relevant Python versions.

3.6.6 1.4.1 (2015-06-29)

- Fix version number (ezarowny)
- Fix choice options, unbreaking nose-cover (aamirtharaj-rpx, jwhitlock)
- Support 1.8 app loading system (dgladkov)
- Support non-ASCII file names (singingwolfboy)
- Better PEP8'd assertion names (roganov)

3.6.7 1.4 (2015-04-23)

- Django 1.8 support (timc3, adepue, jwhitlock)
- Support –testrunner option (st4lk)
- Fix REUSE_DB second run in py3k (edrmp)

3.6.8 1.3 (2014-12-05)

- Django 1.6 and 1.7 support (conrado, co3k, Nepherhotep, mbertheau)
- Python 3.3 and 3.4 testing and support (frewsxcv, jsocol)

3.6.9 1.2 (2013-07-23)

- Python 3 support (melinath and jonashaag)
- Django 1.5 compat (fabiosantoscode)

3.6.10 1.1 (2012-05-19)

- Django TransactionTestCases don't clean up after themselves; they leave junk in the DB and clean it up only on `_pre_setup`. Thus, Django makes sure these tests run last. Now django-nose does, too. This means one fewer source of failures on existing projects. (Erik Rose)
- Add support for hygienic TransactionTestCases. (Erik Rose)
- Support models that are used only for tests. Just put them in any file imported in the course of loading tests. No more crazy hacks necessary. (Erik Rose)

- Make the fixture bundler more conservative, fixing some conceivable situations in which fixtures would not appear as intended if a TransactionTestCase found its way into the middle of a bundle. (Erik Rose)
- Fix an error that would surface when using SQLAlchemy with connection pooling. (Roger Hu)
- Gracefully ignore the new `--liveserver` option introduced in Django 1.4; don't let it through to nose. (Adam DePue)

3.6.11 1.0 (2012-03-12)

- New fixture-bundling plugin for avoiding needless fixture setup (Erik Rose)
- Moved FastFixtureTestCase in from test-utils, so now all the fixture-bundling stuff is in one library. (Erik Rose)
- Added the REUSE_DB setting for faster startup and shutdown. (Erik Rose)
- Fixed a crash when printing options with certain verbosities. (Daniel Abel)
- Broke hard dependency on MySQL. Support PostgreSQL. (Roger Hu)
- Support SQLite, both memory- and disk-based. (Roger Hu and Erik Rose)
- Nail down versions of the package requirements. (Daniel Mizyrycki)

3.6.12 0.1.3 (2010-04-15)

- Even better coverage support (rozza)
- README fixes (carljm and ionelmc)
- optparse OptionGroups are handled better (outofculture)
- nose plugins are loaded before listing options

3.6.13 0.1.2 (2010-08-14)

- run_tests API support (carjm)
- better coverage numbers (rozza & miracle2k)
- support for adding custom nose plugins (kumar303)

3.6.14 0.1.1 (2010-06-01)

- Cleaner installation (Michael Fladischer)

3.6.15 0.1 (2010-05-18)

- Class-based test runner (Antti Kaihola)
- Django 1.2 compatibility (Antti Kaihola)
- Mapping Django verbosity to nose verbosity

3.6.16 0.0.3 (2009-12-31)

- Python 2.4 support (Blake Winton)
- GeoDjango spatial database support (Peter Baumgartner)
- Return the number of failing tests on the command line

3.6.17 0.0.2 (2009-10-01)

- rst readme (Rob Madole)

3.6.18 0.0.1 (2009-10-01)

- birth!

CHAPTER 4

Indices and tables

- genindex
- modindex
- search