django-taggit

Release 1.3.0

May 02, 2022

Contents

`django-taggit` is a reusable Django application designed to make adding tagging to your project easy and fun.

`django-taggit` works with Django 2.2+ and Python 3.6+.

CHAPTER 1

Getting Started

To get started using `django-taggit` simply install it with `pip`:

```
$ pip install django-taggit
```

Add `"taggit"` to your project's `INSTALLED_APPS` setting.

Run *./manage.py migrate*.

And then to any model you want tagging on do the following:

```
from django.db import models

from taggit.managers import TaggableManager

class Food(models.Model):
    # ... fields here

    tags = TaggableManager()
```

Note: If you want `django-taggit` to be **CASE-INSENSITIVE** when looking up existing tags, you'll have to set `TAGGIT_CASE_INSENSITIVE` (in `settings.py` or wherever you have your Django settings) to `True` (`False` by default):

```
TAGGIT_CASE_INSENSITIVE = True
```

1.1 Settings

The following Django-level settings affect the behavior of the library

- `TAGGIT_CASE_INSENSITIVE`

  When set to `True`, tag lookups will be case insensitive. This defaults to `False`.

‘‘ **TAGGIT_STRIP_UNICODE_WHEN_SLUGIFYING** When this is set to `True`, tag slugs will be limited to ASCII characters. In this case, if you also have `` `unidecode `` installed, then tag sluggification will transform a tag like to `ai-ueo`. If you do not have `unidecode` installed, then you will usually be outright stripping unicode, meaning that something like `hello` will be slugified as `hello`.

This value defaults to `False`, meaning that unicode is preserved in slugification.

Because the behavior when `True` is set leads to situations where slugs can be entirely stripped to an empty string, we recommend not activating this.

CHAPTER 2

Tags in forms

The `TaggableManager` will show up automatically as a field in a `ModelForm` or in the admin. Tags input via the form field are parsed as follows:

- If the input doesn't contain any commas or double quotes, it is simply treated as a space-delimited list of tag names.
- If the input does contain either of these characters:
  - Groups of characters which appear between double quotes take precedence as multi-word tags (so double quoted tag names may contain commas). An unclosed double quote will be ignored.
  - Otherwise, if there are any unquoted commas in the input, it will be treated as comma-delimited. If not, it will be treated as space-delimited.

Examples:

Tag input string	Resulting tags	Notes
apple ball cat	`["apple", "ball", "cat"]`	No commas, so space delimited
apple, ball cat	`["apple", "ball cat"]`	Comma present, so comma delimited
"apple, ball" cat dog	`["apple, ball", "cat", "dog"]`	All commas are quoted, so space delimited
"apple, ball", cat dog	`["apple, ball", "cat dog"]`	Contains an unquoted comma, so comma delimited
apple "ball cat" dog	`["apple", "ball cat", "dog"]`	No commas, so space delimited
"apple" "ball dog	`["apple", "ball", "dog"]`	Unclosed double quote is ignored

2.1 `commit=False`

If, when saving a form, you use the `commit=False` option you'll need to call `save_m2m()` on the form after you save the object, just as you would for a form with normal many to many fields on it:

```
if request.method == "POST":
    form = MyFormClass(request.POST)
    if form.is_valid():
        obj = form.save(commit=False)
        obj.user = request.user
        obj.save()
        # Without this next line the tags won't be saved.
        form.save_m2m()
```

You can check the details over in the Django documentation on form saving.

CHAPTER 3

Using tags in the admin

By default if you have a *TaggableManager* on your model it will show up in the admin, just as it will in any other form.

If you are specifying `ModelAdmin.fieldsets`, include the name of the *TaggableManager* as a field:

```
fieldsets = (
    (None, {'fields': ('tags',)}),
)
```

3.1 Including tags in `ModelAdmin.list_display`

One important thing to note is that you *cannot* include a *TaggableManager* in `ModelAdmin.list_display`. If you do you'll see an exception that looks like:

```
AttributeError: '_TaggableManager' object has no attribute 'name'
```

This is for the same reason that you cannot include a `ManyToManyField`: it would result in an unreasonable number of queries being executed.

If you want to show tags in `ModelAdmin.list_display`, you can add a custom display method to the `ModelAdmin`, using `prefetch_related()` to minimize queries:

```
class MyModelAdmin(admin.ModelAdmin):
    list_display = ['tag_list']

    def get_queryset(self, request):
        return super().get_queryset(request).prefetch_related('tags')

    def tag_list(self, obj):
        return u", ".join(o.name for o in obj.tags.all())
```

CHAPTER 4

Usage With Django Rest Framework

Because the tags in *django-taggit* need to be added into a *TaggableManager()* we cannot use the usual *Serializer* that we get from Django REST Framework. Because this is trying to save the tags into a *list*, which will throw an exception.

To accept tags through a *REST* API call we need to add the following to our *Serializer*:

```
from taggit.serializers import (TagListSerializerField,
                                TaggitSerializer)


class YourSerializer(TaggitSerializer, serializers.ModelSerializer):

    tags = TagListSerializerField()

    class Meta:
        model = YourModel
        fields = '__all__'
```

And you're done, so now you can add tags to your model.

CHAPTER 5

The API

After you've got your `TaggableManager` added to your model you can start playing around with the API.

class TaggableManager(*[verbose_name="Tags", help_text="A comma-separated list of tags.", through=None, blank=False]*)

Parameters

- **verbose_name** – The verbose_name for this field.
- **help_text** – The help_text to be used in forms (including the admin).
- **through** – The through model, see *Customizing taggit* for more information.
- **blank** – Controls whether this field is required.

add(**tags*, *through_defaults=None*, *tag_kwargs=None*)

This adds tags to an object. The tags can be either `Tag` instances, or strings:

```
>>> apple.tags.all()
[]
>>> apple.tags.add("red", "green", "fruit")
```

Use the `through_defaults` argument to specify values for your custom `through` model, if needed.

The `tag_kwargs` argument allows one to specify parameters for the tags themselves.

remove(**tags*)

Removes a tag from an object. No exception is raised if the object doesn't have that tag.

clear()

Removes all tags from an object.

set(*tags*, ***, *through_defaults=None*, *clear=False*)

If `clear = True` removes all the current tags and then adds the specified tags to the object. Otherwise sets the object's tags to those specified, removing only the missing tags and adding only the new tags.

Use the `through_defaults` argument to specify values for your custom `through` model, if needed.

similar_objects()
Returns a list (not a lazy QuerySet) of other objects tagged similarly to this one, ordered with most similar first. Each object in the list is decorated with a similar_tags attribute, the number of tags it shares with this object.

If the model is using generic tagging (the default), this method searches tagged objects from all classes. If you are querying on a model with its own tagging through table, only other instances of the same model will be returned.

names()
Convenience method, returning a ValuesListQuerySet (basically just an iterable) containing the name of each tag as a string:

```
>>> apple.tags.names()
[u'green and juicy', u'red']
```

slugs()
Convenience method, returning a ValuesListQuerySet (basically just an iterable) containing the slug of each tag as a string:

```
>>> apple.tags.slugs()
[u'green-and-juicy', u'red']
```

Hint: You can subclass _TaggableManager (note the underscore) to add methods or functionality. TaggableManager takes an optional manager keyword argument for your custom class, like this:

```
class Food(models.Model):
    # ... fields here
    tags = TaggableManager(manager=_CustomTaggableManager)
```

5.1 Filtering

To find all of a model with a specific tags you can filter, using the normal Django ORM API. For example if you had a Food model, whose TaggableManager was named tags, you could find all the delicious fruit like so:

```
>>> Food.objects.filter(tags__name__in=["delicious"])
[<Food: apple>, <Food: pear>, <Food: plum>]
```

If you're filtering on multiple tags, it's very common to get duplicate results, because of the way relational databases work. Often you'll want to make use of the distinct() method on QuerySets:

```
>>> Food.objects.filter(tags__name__in=["delicious", "red"])
[<Food: apple>, <Food: apple>]
>>> Food.objects.filter(tags__name__in=["delicious", "red"]).distinct()
[<Food: apple>]
```

You can also filter by the slug on tags. If you're using a custom Tag model you can use this API to filter on any fields it has.

CHAPTER 6

Frequently Asked Questions

- How can I get all my tags?

  If you are using just an out-of-the-box setup, your tags are storred in the *Tag* model (found in *taggit.models*). If this is a custom model (for example you have your own models derived from *ItemBase*), then you'll need to query that one instead.

  So if you are using the standard setup, `Tag.objects.all()` will give you the tags.

  - How can I use this with factory_boy?

  Since these are all built off of many-to-many relationships, you can check out factory_boy's documentation on this topic and get some ideas on how to deal with tags.

  One way to handle this is with post-generation hooks:

```
class ProductFactory(DjangoModelFactory):
     # Rest of the stuff

     @post_generation
     def tags(self, create, extracted, **kwargs):
         if not create:
             return

         if extracted:
             self.tags.add(*extracted)
```

CHAPTER 7

Customizing taggit

7.1 Using a Custom Tag or Through Model

By default `django-taggit` uses a "through model" with a `GenericForeignKey` on it, that has another `ForeignKey` to an included `Tag` model. However, there are some cases where this isn't desirable, for example if you want the speed and referential guarantees of a real `ForeignKey`, if you have a model with a non-integer primary key, or if you want to store additional data about a tag, such as whether it is official. In these cases `django-taggit` makes it easy to substitute your own through model, or `Tag` model.

Note: Including 'taggit' in `settings.py` INSTALLED_APPS list will create the default `django-taggit` and "through model" models. If you would like to use your own models, you will need to remove 'taggit' from `settings.py`'s INSTALLED_APPS list.

To change the behavior there are a number of classes you can subclass to obtain different behavior:

Class name	Behavior
`TaggedItemBase`	Allows custom `ForeignKeys` to models.
`GenericTaggedItemBase`	Allows custom `Tag` models. Tagged models use an integer primary key.
`GenericUUIDTaggedItemBase`	Allows custom `Tag` models. Tagged models use a UUID primary key.
`CommonGenericTaggedItemBase`	Allows custom `Tag` models and `GenericForeignKeys` to models.
`ItemBase`	Allows custom `Tag` models and `ForeignKeys` to models.

7.1.1 Custom ForeignKeys

Your intermediary model must be a subclass of `taggit.models.TaggedItemBase` with a foreign key to your content model named `content_object`. Pass this intermediary model as the `through` argument to `TaggableManager`:

```
from django.db import models

from taggit.managers import TaggableManager
```

(continues on next page)

(continued from previous page)

```
from taggit.models import TaggedItemBase


class TaggedFood(TaggedItemBase):
    content_object = models.ForeignKey('Food', on_delete=models.CASCADE)

class Food(models.Model):
    # ... fields here

    tags = TaggableManager(through=TaggedFood)
```

Once this is done, the API works the same as for GFK-tagged models.

7.1.2 Custom GenericForeignKeys

The default `GenericForeignKey` used by `django-taggit` assume your tagged object use an integer primary key. For non-integer primary key, your intermediary model must be a subclass of `taggit.models.CommonGenericTaggedItemBase` with a field named `"object_id"` of the type of your primary key.

For example, if your primary key is a string:

```
from django.db import models

from taggit.managers import TaggableManager
from taggit.models import CommonGenericTaggedItemBase, TaggedItemBase

class GenericStringTaggedItem(CommonGenericTaggedItemBase, TaggedItemBase):
    object_id = models.CharField(max_length=50, verbose_name=_('Object id'),␣
↪db_index=True)

class Food(models.Model):
    food_id = models.CharField(primary_key=True)
    # ... fields here

    tags = TaggableManager(through=GenericStringTaggedItem)
```

7.1.3 GenericUUIDTaggedItemBase

A common use case of a non-integer primary key, is UUID primary key. `django-taggit` provides a base class `GenericUUIDTaggedItemBase` ready to use with models using an UUID primary key:

```
from django.db import models
from django.utils.translation import gettext_lazy as _

from taggit.managers import TaggableManager
from taggit.models import GenericUUIDTaggedItemBase, TaggedItemBase

class UUIDTaggedItem(GenericUUIDTaggedItemBase, TaggedItemBase):
    # If you only inherit GenericUUIDTaggedItemBase, you need to define
    # a tag field. e.g.
    # tag = models.ForeignKey(Tag, related_name="uuid_tagged_items", on_
↪delete=models.CASCADE)

    class Meta:
```

(continues on next page)

(continued from previous page)

```
        verbose_name = _("Tag")
        verbose_name_plural = _("Tags")

class Food(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4,␣
↪editable=False)
    # ... fields here

    tags = TaggableManager(through=UUIDTaggedItem)
```

7.1.4 Custom tag

When providing a custom `Tag` model it should be a `ForeignKey` to your tag model named `"tag"`. If your custom `Tag` model has extra parameters you want to initialize during setup, you can do so by passing it along via the `tag_kwargs` parameter of `TaggableManager.add`. For example `my_food.tags.add("tag_name1", "tag_name2", tag_kwargs={"my_field":3})`:

```
from django.db import models
from django.utils.translation import gettext_lazy as _

from taggit.managers import TaggableManager
from taggit.models import TagBase, GenericTaggedItemBase


class MyCustomTag(TagBase):
    # ... fields here

    class Meta:
        verbose_name = _("Tag")
        verbose_name_plural = _("Tags")

    # ... methods (if any) here


class TaggedWhatever(GenericTaggedItemBase):
    # TaggedWhatever can also extend TaggedItemBase or a combination of
    # both TaggedItemBase and GenericTaggedItemBase. GenericTaggedItemBase
    # allows using the same tag for different kinds of objects, in this
    # example Food and Drink.

    # Here is where you provide your custom Tag class.
    tag = models.ForeignKey(
        MyCustomTag,
        on_delete=models.CASCADE,
        related_name="%(app_label)s_%(class)s_items",
    )


class Food(models.Model):
    # ... fields here

    tags = TaggableManager(through=TaggedWhatever)


class Drink(models.Model):
```

(continues on next page)

(continued from previous page)

```
    # ... fields here

    tags = TaggableManager(through=TaggedWhatever)
```

class TagBase

slugify (*tag*, *i=None*)
By default `taggit` uses `django.utils.text.slugify()` to calculate a slug for a given tag. However, if you want to implement your own logic you can override this method, which receives the `tag` (a string), and `i`, which is either `None` or an integer, which signifies how many times the slug for this tag has been attempted to be calculated, it is `None` on the first time, and the counting begins at `1` thereafter.

7.2 Using a custom tag string parser

By default `django-taggit` uses `taggit.utils._parse_tags` which accepts a string which may contain one or more tags and returns a list of tag names. This parser is quite intelligent and can handle a number of edge cases; however, you may wish to provide your own parser for various reasons (e.g. you can do some preprocessing on the tags so that they are converted to lowercase, reject certain tags, disallow certain characters, split only on commas rather than commas and whitespace, etc.). To provide your own parser, write a function that takes a tag string and returns a list of tag names. For example, a simple function to split on comma and convert to lowercase:

```
def comma_splitter(tag_string):
    return [t.strip().lower() for t in tag_string.split(',') if t.strip()]
```

You need to tell `taggit` to use this function instead of the default by adding a new setting, `TAGGIT_TAGS_FROM_STRING` and providing it with the dotted path to your function. Likewise, you can provide a function to convert a list of tags to a string representation and use the setting `TAGGIT_STRING_FROM_TAGS` to override the default value (which is `taggit.utils._edit_string_for_tags`):

```
def comma_joiner(tags):
    return ', '.join(t.name for t in tags)
```

If the functions above were defined in a module, `appname.utils`, then your project settings.py file should contain the following:

```
TAGGIT_TAGS_FROM_STRING = 'appname.utils.comma_splitter'
TAGGIT_STRING_FROM_TAGS = 'appname.utils.comma_joiner'
```

CHAPTER 8

Contributing to django-taggit

This is a Jazzband project. By contributing you agree to abide by the Contributor Code of Conduct and follow the guidelines.

Thank you for taking the time to contribute to django-taggit.

Follow these guidelines to speed up the process.

8.1 Reach out before you start

Before opening a new issue, look if somebody else has already started working on the same issue in the GitHub issues and pull requests.

8.2 Fork the repository

Once you have forked this repository to your own GitHub account, install your own fork in your development environment:

```
git clone git@github.com:<your_fork>/django-taggit.git
cd django-taggit
python setup.py develop
```

8.3 Running tests

django-taggit uses tox to run tests:

```
tox
```

8.4 Follow style conventions (black, flake8, isort)

Check that your changes are not breaking the style conventions with:

```
tox -e black,flake8,isort
```

8.5 Update the documentation

If you introduce new features or change existing documented behavior, please remember to update the documentation.

The documentation is located in the `docs` directory of the repository.

To do work on the docs, proceed with the following steps:

```
pip install sphinx
sphinx-build -n -W docs docs/_build
```

8.6 Send pull request

It is now time to push your changes to GitHub and open a pull request!

8.7 Release Checklist

To make a release, the following needs to happen:

- Bump the version number in `taggit/__init__.py`
- Update the changelog (making sure to add the (Unreleased) section to the top)
- Get those changes onto the `master` branch
- Tag the commit with the version number
- CI should then upload a release to be verified through Jazzband

CHAPTER 9

External Applications

In addition to the features included in `django-taggit` directly, there are a number of external applications which provide additional features that may be of interest.

Note: Despite their mention here, the following applications are in no way official, nor have they in any way been reviewed or tested.

If you have an application that you'd like to see listed here, simply fork django-taggit on github, add it to this list, and send a pull request.

- django-taggit-anywhere: Simpler approach to tagging with `taggit`. Additionally this project provides easy-to-use integration with `django-taggit-helpers` and `django-taggit-labels`.
- django-taggit-helpers: Makes it easier to work with admin pages of models associated with `taggit` tags by adding helper classes: `TaggitCounter`, `TaggitListFilter`, `TaggitStackedInline`, `TaggitTabularInline`.
- django-taggit-labels: Provides a clickable label widget for the Django admin for user friendly selection from managed tag sets.
- django-taggit-serializer: Adds functionality for using `taggit` with `django-rest-framework`.
- django-taggit-suggest: Provides support for defining keyword and regular expression rules for suggesting new tags for content. This used to be available at `taggit.contrib.suggest`.
- django-taggit-templatetags: Provides several templatetags, including one for tag clouds, to expose various `taggit` APIs directly to templates.
- django-taggit-bulk: An admin action for the bulk tagging from the model admin instance list view.

CHAPTER 10

Changelog

10.1 (Unreleased)

10.2 3.0.0 (2022-05-02)

- **Backwards incompatible:** Tag slugification used to silently strip non-ASCII characters from the tag name to make the slug. This leads to a lot of confusion for anyone using languages with non-latin alphabets, as well as weird performance issues.

  Tag slugification will now, by default, maintain unicode characters as-is during slugification. This will lead to less surprises, but might cause issues for you if you are expecting all of your tag slugs to fit within a regex like `[a-zA-Z0-9]` (for example in URL routing configurations).

  Generally speaking, this should not require action on your part as a library user, as existing tag slugs are persisted in the database, and only new tags will receive the enhanced unicode-compatible slug.

  If you wish to maintain the old stripping behavior, set the setting `TAGGIT_STRIP_UNICODE_WHEN_SLUGIFYING` to `True`.

  As a reminder, custom tag models can easily customize slugification behavior by overriding the `slugify` method to your business needs.

‘‘ Drop Django 2.2 support.

10.3 2.1.0 (2022-01-24)

- Add Python 3.10 support.
- Add Django 4.0 support.
- Drop Django 3.1 support.

10.4 2.0.0 (2021-11-14)

- **Backwards incompatible:** `TaggableManager.set` now takes a list of tags (instead of varargs) so that its API matches Django's `RelatedManager.set`. Example:
  - previously: `item.tags.set("red", "blue")`
  - now: `item.tags.set(["red", "blue"])`
- Fix issue where `TagField` would incorrectly report that a field has changed on empty values.
- Update Russian translation.
- Add Persian translation
- Fix issue for many languages where content types were not being properly translated.
- Provide translators additional context regarding strings in TagBase model.

10.5 1.5.1 (2021-07-01)

- Fix compiled Ukranian translation (which would cause a failure on load for this locale).
- Update compiled Danish translation.

10.6 1.5.0 (2021-06-30)

- Vendor in the *django-taggit-serializer* project (under *taggit.serializers*).
- Add Arabic translation.
- Add Ukranian translation.

10.7 1.4.0 (2021-04-19)

- Add Python 3.9 support.
- Remove Python 3.5 support.
- Add Django 3.2 support.
- Remove Django 1.11 and 3.0 support.
- Add Danish translation.
- Fix crashing that could occur with `similar_objects` in multi-inheritance contexts.
- Add support for custom fields on through table models with *through_defaults* for `TaggedManager.add` and `TaggedManager.set`.

10.8 1.3.0 (2020-05-19)

- Model and field `verbose_name` and `verbose_name_plural` attributes are now lowercase. This simplifies using the name in the middle of a sentence. When used as a header, title, or at the beginning of a sentence, a text transformed can be used to adjust the case.

- Fix prefetch_related when using UUIDTaggedItem.
- Allow for passing in extra constructor parameters when using `TaggableManager.add`. This is especially useful when using custom tag models.

10.9 1.2.0 (2019-12-03)

- **Removed** support for end-of-life Django 2.0 and 2.1.
- Added support for Django 3.0.
- Added support for Python 3.8.
- Moved `TaggedItemBase.tags_for()` to ItemBase.
- Replaced reference to removed Django's `.virtual_fields` with `.private_field`.
- Added `TextareaTagWidget`.

10.10 1.1.0 (2019-03-22)

- Added Finnish translation.
- Updated Chinese translation.
- Updated Esperanto translation.
- Fix `form.changed_data` to allow early access for a tags defined with `blank=True`.

10.11 1.0.0 (2019-03-17)

- **Backwards incompatible:** Remove support for Python 2.
- Added `has_changed()` method to `taggit.forms.TagField`.
- Added multi-column unique constraint to model `TaggedItem` on fields `content_type`, `object_id`, and `tag`. Databases that contain duplicates will need to add a data migration to resolve these duplicates.
- Fixed `TaggableManager.most_common()` to always evaluate lazily. Allows placing a `.most_common()` query at the top level of a module.
- Fixed setting the `related_name` on a tags manager that exists on a model named `Name`.

10.12 0.24.0 (2019-02-19)

- The project has moved to Jazzband. This is the first release under the new organization. The new repository URL is https://github.com/jazzband/django-taggit.
- Added support for Django 2.2.
- Fixed a race condition in `TaggableManager`.
- Removed method `ItemBase.bulk_lookup_kwargs()`.
- Fixed view `tagged_object_list` to set `queryset.model` as `ListView.model` (was previously set as a `ContentType` instance).

- `_TaggableManager` and `TaggableManager` now always call the parent class `__init__`.
- Removed `TaggableRel` and replaced uses with `ManyToManyRel`.

10.13 0.23.0 (2018-08-07)

- **Backwards incompatible:** Remove support for Django < 1.11
- Added support for Django 2.1 and Python 3.7
- Moved TagWidget value conversion from TagWidget.render() to TagWidget.format_value()

10.14 0.22.2 (2017-12-27)

- Added support for Django 2.0
- **Backwards incompatible:** Dropped support for EOL Python 3.3

10.15 0.22.1 (2017-04-22)

- Update spanish translation
- Add testing for Django 1.11 and Python 3.6
- introduce isort and flake8 in the CI
- [docs] Fixed links to external apps
- Improved auto-slug in TagBase to support UUID pk
- [docs] Added contribution guidelines

10.16 0.22.0 (2017-01-29)

- **Backwards incompatible:** Drop support for Django 1.7

10.17 0.21.6 (2017-01-25)

- Fix case-insensitive tag creation when setting to a mix of new and existing tags are used

10.18 0.21.5 (2017-01-21)

- Check for case-insensitive duplicates when creating new tags

10.19 0.21.4 (2017-01-10)

- Support __gt__ and __lt__ ordering on Tags

10.20 0.21.3 (2016-10-07)

- Fix list view

10.21 0.21.2 (2016-08-31)

- Update Python version classifiers in setup.py
- Add Greek translation

10.22 0.21.1 (2016-08-25)

- Document supported versions of Django; fix Travis to test these versions.

10.23 0.21.0 (2016-08-22)

- Fix form tests on Django 1.10
- Address list_display and fieldsets in admin docs
- external_apps.txt improvements
- Remove support for Django 1.4-1.6, again.

10.24 0.20.2 (2016-07-11)

- Add extra_filters argument to the manager's most_common method

10.25 0.20.1 (2016-06-23)

- Specify *app_label* for *Tag* and *TaggedItem*

10.26 0.20.0 (2016-06-19)

- Fix UnboundLocalError in _TaggableManager.set(..)
- Update doc links to reflect RTD domain changes
- Improve Russian translations

10.27 0.19.1 (2016-05-25)

- Add app config, add simplified Chinese translation file

10.28 0.19.0 (2016-05-23)

- Implementation of m2m_changed signal sending
- Code and tooling improvements

10.29 0.18.3 (2016-05-12)

- Added Spanish and Turkish translations

10.30 0.18.2 (2016-05-08)

- Add the min_count parameter to managers.most_common function

10.31 0.18.1 (2016-03-30)

- Address deprecation warnings

10.32 0.18.0 (2016-01-18)

- Add option to override default tag string parsing
- Drop support for Python 2.6

10.33 0.17.6 (2015-12-09)

- Silence Django 1.9 warning

10.34 0.17.5 (2015-11-27)

- Django 1.9 compatibility fix

10.35 0.17.4 (2015-11-25)

- Allows custom Through Model with GenericForeignKey

10.36 0.17.3 (2015-10-26)

- Silence Django 1.9 warning about on_delete

10.37 0.17.2 (2015-10-25)

- Django 1.9 beta compatibility

10.38 0.17.1 (2015-09-10)

- Fix unknown column *object_id* issue with Django 1.6+

10.39 0.17.0 (2015-08-14)

- Database index added on TaggedItem fields content_type & object_id

10.40 0.16.4 (2015-08-13)

- Access default manager via class instead of instance

10.41 0.16.3 (2015-08-08)

- Prevent IntegrityError with custom TagBase classes

10.42 0.16.2 (2015-07-13)

- Fix an admin bug related to the *Manager* property *through_fields*

10.43 0.16.1 (2015-07-09)

- Fix bug that assumed all primary keys are named 'id'

10.44 0.16.0 (2015-07-04)

- Add option to allow case-insensitive tags

10.45 0.15.0 (2015-06-23)

- Fix wrong slugs for non-latin chars. Only works if optional GPL dependency (unidecode) is installed.

10.46 0.14.0 (2015-04-26)

- Prevent extra JOIN when prefetching
- Prevent _meta warnings with Django 1.8

10.47 0.13.0 (2015-04-02)

- Django 1.8 support

10.48 0.12.3 (2015-03-03)

- Specify that the internal type of the TaggitManager is a ManyToManyField

10.49 0.12.2 (2014-21-09)

- Fixed 1.7 migrations.

10.50 0.12.1 (2014-10-08)

- Final (hopefully) fixes for the upcoming Django 1.7 release.
- Added Japanese translation.

10.51 0.12.0 (2014-20-04)

- **Backwards incompatible:** Support for Django 1.7 migrations. South users have to set `SOUTH_MIGRATION_MODULES` to use `taggit.south_migrations` for taggit.
- **Backwards incompatible:** Django's new transaction handling is used on Django 1.6 and newer.
- **Backwards incompatible:** `Tag.save` got changed to opportunistically try to save the tag and if that fails fall back to selecting existing similar tags and retry – if that fails too an `IntegrityError` is raised by the database, your app will have to handle that.
- Added Italian and Esperanto translations.

10.52 0.11.2 (2013-13-12)

- Forbid multiple TaggableManagers via generic foreign keys.

10.53 0.11.1 (2013-25-11)

- Fixed support for Django 1.4 and 1.5.

10.54 0.11.0 (2013-25-11)

- Added support for prefetch_related on tags fields.
- Fixed support for Django 1.7.
- Made the tagging relations unserializeable again.
- **Allow more than one TaggableManager on models (assuming concrete FKs are** used for the relations).

10.55 0.10.0 (2013-17-08)

- Support for Django 1.6 and 1.7.
- Python3 support
- **Backwards incompatible:** Dropped support for Django < 1.4.5.
- Tag names are unique now, use the provided South migrations to upgrade.

10.56 0.9.2 (2011-01-17)

- **Backwards incompatible:** Forms containing a *`TaggableManager`* by default now require tags, to change this provide `blank=True` to the *`TaggableManager`*.
- Now works with Django 1.3 (as of beta-1).

10.57 0.9.0 (2010-09-22)

- Added a Hebrew locale.
- Added an index on the `object_id` field of `TaggedItem`.
- When displaying tags always join them with commas, never spaces.
- The docs are now available online.
- Custom `Tag` models are now allowed.
- **Backwards incompatible:** Filtering on tags is no longer `filter(tags__in=["foo"])`, it is written `filter(tags__name__in=["foo"])`.
- Added a German locale.
- Added a Dutch locale.
- **Removed `taggit.contrib.suggest`, it now lives in an external application,** see *External Applications* for more information.

10.58 0.8.0 (2010-06-22)

- Fixed querying for objects using `exclude(tags__in=tags)`.
- Marked strings as translatable.

- Added a Russian translation.
- Created a mailing list.
- Smarter tagstring parsing for form field; ported from Jonathan Buchanan's django-tagging. Now supports tags containing commas. See *Tags in forms* for details.
- Switched to using savepoints around the slug generation for tags. This ensures that it works fine on databases (such as Postgres) which dirty a transaction with an `IntegrityError`.
- Added Python 2.4 compatibility.
- Added Django 1.1 compatibility.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Index