

---

# **django-werewolf Documentation**

***Release 0.1***

**Artur Barseghyan <[artur.barseghyan@gmail.com](mailto:artur.barseghyan@gmail.com)>**

November 29, 2013



---

# Contents

---



django-werewolf



---

# Description

---

Item publishing workflow for Django (fully integrated into Django admin).





---

# Installation

---

1. Install django-werewolf into your virtual environment:

\$ pip install django-werewolf

2. Add *werewolf* to your `INSTALLED_APPS`.

That's all. See the *Usage and examples* section for more.



---

## Usage and examples

---

It's all about item publishing in a workflow. We have various *intermediate* statuses (work in-progress) and a final *status* which means that the item is actually published. Some users should be able to set the item status to *published*, some others not. This app allows you (and gives you a good working example with pre-configured django environment) to write a custom workflow for publishing your items with minimal efforts.

For a complete example of a working django-werewolf app see the <https://bitbucket.org/barseghyanartur/django-werewolf> (example directory) and read the *readme.rst* of the *news* app.

### 3.1 settings.py

```
>>> # Workflow statuses; order is preserved.
>>> WEREWOLF_STATUS_CHOICES = (
>>>     ('new', gettext('New')), # New - this is how it's assigned to a writer.
>>>     ('draft', gettext('Draft')), # Draft - this is how the writer works on it.
>>>     ('ready', gettext('Ready')), # Ready to be reviewed by editor.
>>>     ('reviewed', gettext('Reviewed')), # Reviewed by editor (means positive and ready to be publ.
>>>     ('published', gettext('Published')), # Published.
>>> )
>>>
>>> # Published status.
>>> WEREWOLF_STATUS_PUBLISHED = 'published'
>>>
>>> # When set to True, django-reversion is used.
>>> WEREWOLF_USE_DJANGO_REVERSION = True
```

### 3.2 news/models.py

In the example below we have a basic news item model. We have Chief Editors with full access to news items, we have editors with less privileges and Writers with very little privileges. Chief Editors create articles, select an Editor and a Writer (both get notified) and let them work on the article. Writers can only set an article status to *new*, *draft* and *ready* (ready to be checked). Editors review the articles with status *ready* and set the status to *reviewed*. Chief Editors publish articles that are *reviewed*. Your implementation can be as custom as you want it. Think in Django user groups (`django.contrib.auth.models.Group`) and Django permissions system.

NOTE: See the *Permission tuning* section.

```
>>> from django.contrib.auth.models import User
>>>
>>> from werewolf.models import WerewolfBaseModel, WerewolfBaseMeta
>>>
>>> _chief_editors = {'groups__name__iexact': 'Chief editors'}
>>> _editors = {'groups__name__iexact': 'Editors'}
>>> _writers = {'groups__name__iexact': 'Writers'}
>>>
>>> class NewsItem(WerewolfBaseModel): # Important!
>>>     title = models.CharField(_("Title"), max_length=100)
>>>     body = models.TextField(_("Body"))
>>>     date_published = models.DateTimeField(_("Date published"), default=datetime.datetime.now())
>>>     author = models.ForeignKey(User, verbose_name=_("Author"), related_name='authors', limit_choices_to=_editors)
>>>     editor = models.ForeignKey(User, verbose_name=_("Editor"), related_name='editors', limit_choices_to=_writers)
>>>     chief_editor = models.ForeignKey(User, verbose_name=_("Chief editor"), related_name='chief_editors', limit_choices_to=_chief_editors)
>>>
>>>     class Meta(WerewolfBaseMeta): # Important!
>>>         verbose_name = "News item"
>>>         verbose_name_plural = "News items"
```

Or if you want to define custom permissions for your model as well, do extend the werewolf permissions as follows:

```
>>> from werewolf.models import WerewolfBaseModel
>>> from werewolf.utils import extend_werewolf_permissions
>>>
>>> class NewsItem(WerewolfBaseModel):
>>>     # Your fields here
>>>     class Meta:
>>>         verbose_name = "News item"
>>>         verbose_name_plural = "News items"
>>>
>>>     # Important!
>>>     permissions = extend_werewolf_permissions(
>>>         ('can_change_author', _("Can change author")),
>>>         ('can_change_editor', _("Can change editor")),
>>>         ('can_change_chief_editor', _("Can change chief editor"))
>>>     )
```

## 3.3 news/admin.py

Basic admin for the news item model.

NOTE: See the *Permission tuning* section.

```
>>> from werewolf.admin import WerewolfBaseAdmin
>>>
>>> from news.models import NewsItem
>>>
>>> class NewsItemAdmin(WerewolfBaseAdmin):
>>>     # Your code comes here
>>>
>>> admin.site.register(NewsItem, NewsItemAdmin)
```

NOTE: If you override the `queryset` method of your model's admin class, make sure to see the source code of `werewolf.admin.WerewolfBaseAdmin.queryset` and copy the approach from there. Otherwise, your users with no per-

mission to change the *published* status will be able to change the status of already published items to non-published statuses.

### 3.4 news/views.py

```
>>> from news.models import NewsItem
>>>
>>> def browse(request):
>>>     news_items = NewsItem._default_manager.published()
>>>     # Other code
```

### 3.5 news/werewolf\_triggers.py

In order to perform extra tasks on status change, triggers are used. You simply make a new file in your app called *werewolf\_triggers.py* and define custom classes that should be called when a *status* field of your model changes to a certain value. Each trigger should subclass the *werewolf.triggers.WerewolfBaseTrigger* class.

```
>>> from werewolf.triggers import WerewolfBaseTrigger, registry
>>>
>>> class StatusNewTrigger(WerewolfBaseTrigger):
>>>     """
>>>     News item status changed to 'new'.
>>>     """
>>>     def process(self):
>>>         # Your code
>>>
>>> class StatusReadyTrigger(WerewolfBaseTrigger):
>>>     """
>>>     News item status changed to 'ready' (ready for review).
>>>     """
>>>     def process(self):
>>>         # Your code
>>>
>>> # Triggers status change to 'new' for news.newsitem model.
>>> registry.register('news', 'newsitem', 'new', StatusNewTrigger)
>>>
>>> # Triggers status change to 'ready' for news.newsitem model.
>>> registry.register('news', 'newsitem', 'ready', StatusReadyTrigger)
```

### 3.6 urls.py

In order to have triggers autodiscovered, place the following code into your main *urls* module.

```
>>> from werewolf import autodiscover as werewolf_autodiscover
>>> werewolf_autodiscover()
```

### 3.7 Permission tuning

Have in mind our *news.models.NewsItem* model.

1. Create three user groups:

(a) Chief editors (permissions listed):

- news | News item | Can add News item
- news | News item | Can change author
- news | News item | Can change chief editor
- news | News item | Can change editor
- news | News item | Can change News item
- news | News item | Can change status to draft
- news | News item | Can change status to new
- news | News item | Can change status to published
- news | News item | Can change status to ready
- news | News item | Can change status to reviewed
- news | News item | Can delete News item

(a) Editors (permissions listed):

- news | News item | Can change News item
- news | News item | Can change author
- news | News item | Can change status to draft
- news | News item | Can change status to new
- news | News item | Can change status to ready
- news | News item | Can change status to reviewed

(a) Writers (permissions listed):

- news | News item | Can change News item
- news | News item | Can change status to draft
- news | News item | Can change status to new
- news | News item | Can change status to ready

3. Create three users:

- chief editor: Belongs to group *Chief editors*.
- editor: Belongs to group *Editors*.
- writer: Belongs to group *Writers*.

4. Now log into the admin with different user and see your admin for the *News item* (created items with *chiefeditor* account, then view them with *editor* and *writer*).

That's it. If somehow you don't see the new permissions (*Can change status to draft*, *Can change status to new*, etc) run a management command `syncww`:

```
$ ./manage.py syncww
```

---

# Running the example project

---

A working example of a django-werewolf app is available here: <https://bitbucket.org/barseghyanartur/django-werewolf> (see the *example* directory).

1. Go to example/example directory  
`$ cd example/example`
2. Install requirements (in your virtual environment)  
`$ pip install -r ../requirements.txt`
3. Copy local\_settings.example to local\_settings.py  
`$ cp local_settings.example local_settings.py`
4. Create the database  
`$ ./manage.py syncdb`
5. Run the project  
`$ ./manage.py runserver`





---

# License

---

GPL 2.0/LGPL 2.1



---

# Documentation

---

Contents:

## 6.1 werewolf Package

### 6.1.1 werewolf Package

`werewolf.__init__.autodiscover()`

Autodiscovers the werewolf in project apps. Each report file which should be found by werewolf, should be named “werewolf\_triggers.py”.

### 6.1.2 admin Module

**class** `werewolf.admin.WerewolfBaseAdmin(*args, **kwargs)`

Bases: `reversion.admin.VersionAdmin`

Base werewolf admin model.

**formfield\_for\_dbfield** (*db\_field*, *\*\*kwargs*)

Here we replace the choices based on the user permissions.

**media**

**queryset** (*request*)

Make sure users with no rights to edit an object with status, don’t even see it.

**save\_model** (*request*, *obj*, *form*, *change*)

**status\_change\_trigger** (*request*, *obj*, *form*, *change*)

Status change trigger. Executes appropriate registered trigger if applicable.

#### Parameters

- **request** (*django.http.HttpRequest*) –
- **obj** (*django.db.models.Model*) – Subclass of `django.db.models.Model`.
- **form** –
- **change** (*bool*) –

### 6.1.3 helpers Module

`werewolf.helpers.admin_edit_url` (*app\_label*, *module\_name*, *object\_id*, *url\_title=None*)

Gets an admin edit URL for the object given.

#### Parameters

- **app\_label** (*str*) –
- **module\_name** (*str*) –
- **object\_id** (*int*) –
- **url\_title** (*str*) – If given, an HTML a tag is returned with *url\_title* as the tag title. If left to None just the URL string is returned.

#### Return str

`werewolf.helpers.admin_edit_url_for_object` (*obj*, *url\_title=None*)

Gets an admin edit URL for the object given.

#### Parameters

- **obj** (*django.db.models.Model*) – Django model subclass.
- **url\_title** (*str*) – If given, an HTML a tag is returned with *url\_title* as the tag title. If left to None just the URL string is returned.

#### Return str

### 6.1.4 triggers Module

`class werewolf.triggers.WerewolfBaseTrigger` (*obj*, *request*)

Bases: `object`

Werewolf base trigger.

### 6.1.5 utils Module

`werewolf.utils.permission_key` (*status*, *choice\_key*)

Gets the permission key from *choice\_key* given.

#### Parameters

- **status** (*str*) –
- **choice\_key** (*str*) –

#### Return str

`werewolf.utils.permissions_for_base_model` (*permissions=[]*)

Gets/extends permissions for the base model based on the `STATUS_CHOICES` defined.

**Parameters** **permissions** (*list/tuple*) – Permissions you want to have in your model. Those permissions would be extended by werewolf permissions.

#### Return list

`werewolf.utils.status_choices_for_user` (*user*, *app\_label*)

Gets available status choices for the user given.

#### Parameters

- **user** (*django.contrib.auth.models.User*) – User for who the permissions are checked.
- **module\_name** (*str*) – *app\_label* of the model to check permissions to.

**Return list** List of choices in a same form as `werewolf.defaults.STATUS_CHOICES` but then limited to actual choices that user has permissions to.

`werewolf.utils.extend_werewolf_permissions(*args)`

Extends model permissions with werewolf permissions.

#### Example

```
>>> from werewolf.models import WerewolfBaseModel
>>> from werewolf.utils import extend_werewolf_permissions
>>> class NewsItem(WerewolfBaseModel):
>>>     # Some fields here
>>>
>>>     class Meta:
>>>         verbose_name = _("News item")
>>>         verbose_name_plural = _("News items")
>>>
>>>         permissions = extend_werewolf_permissions(
>>>             ('can_change_author', _("Can change author")),
>>>             ('can_change_editor', _("Can change editor")),
>>>             ('can_change_chief_editor', _("Can change chief editor"))
>>>         )
```

## 6.1.6 Subpackages

### management Package

#### Subpackages

#### commands Package

##### syncww Module

**class** `werewolf.management.commands.syncww.Command`  
 Bases: `django.core.management.base.BaseCommand`

**args** = '<app app ...>'

**handle** (*\*args, \*\*options*)

**help** = 'reloads permissions for specified apps, or all apps if no args are specified'

### models Package

#### models Package

**class** `werewolf.models.WerewolfBaseMeta`

Bases: `object`

Base Meta class of the `WerewolfBaseModel`. Every subclass of the `WerewolfBaseModel` shall extend it:

```
>>> from werewolf.models import WerewolfBaseModel, WerewolfBaseMeta
>>> class NewsItem(WerewolfBaseModel): # Important!
>>>     # Your fields here
>>>     class Meta(WerewolfBaseMeta): # Important!
>>>         verbose_name = "News item"
>>>         verbose_name_plural = "News items"
```

Alternatively you can add the permissions attribute:

```
>>> from werewolf.utils import extend_werewolf_permissions
>>> class NewsItem(WerewolfBaseModel):
>>>     # Your fields here
>>>     class Meta:
>>>         verbose_name = "News item"
>>>         verbose_name_plural = "News items"
>>>         permissions = extend_werewolf_permissions(
>>>             ('can_change_author', _("Can change author")),
>>>             ('can_change_editor', _("Can change editor")),
>>>         )
```

**class** werewolf.models.**WerewolfBaseModel** (\*args, \*\*kwargs)

Bases: django.db.models.base.Model

Base Werewolf model. If you want to have a workflow in your model (for statuses like new, draft, published, etc) you should extend this model.

## managers Module

**class** werewolf.models.managers.**WerewolfBaseManager**

Bases: django.db.models.manager.Manager

Werewolf base manager.

---

# Indices and tables

---

- *genindex*
- *modindex*
- *search*





---

# Support

---

For any issues contact me at the e-mail given in the *Author* section.



---

# Author

---

Artur Barseghyan <[artur.barseghyan@gmail.com](mailto:artur.barseghyan@gmail.com)>



---

# Python Module Index

---

## W

werewolf.\_\_init\_\_, ??  
werewolf.admin, ??  
werewolf.helpers, ??  
werewolf.management.commands.syncww, ??  
werewolf.models, ??  
werewolf.models.managers, ??  
werewolf.triggers, ??  
werewolf.utils, ??