Django Admin CLI Documentation

Release 0.1.1

ZuluPro (Anthony Monthe)

July 10, 2015

Contents

If you was writing yourself Django custom commands for make CRUD operation, this app is for you. Django Admin CLI helps you to manage your models in command-line interface as if you where in Django Admimn Site.

Contents:

CHAPTER 1

Setup

1.1 Installation

Install from PyPI with `pip`:

```
pip install django-admin-cli
```

To use `django-admin-cli` in your Django project, add `admin_cli` in your `INSTALLED_APPS` setting.

1.2 Dependencies

`django-admin-cli` supports Django <=1.6 on Python 2.7, <=3.2, 3.2, and 3.3.

CHAPTER 2

Features

This app is based on Django Admin Site and ModelAdmin defined by developpers. It is supposed to allow user to make same things as in Admin site:

- List model's instance:
  - Filtering with Django's Lookup
  - Choosing which field you want including ModelAdmin and Model attributes
  - Default display is the Admin one
- Add an instance:
  - Prepopulate with default values
- Update instances:
  - Filtering with Django's Lookup
- Delete instances:
  - Filtering with Django's Lookup
- Describe model and modeladmin
- System user restriction (Read/Write)
- Use admin actions (further)

Actions

As a Django command, Admin CLI is launched by

```
./manage.py cli <model_name> <action>
```

All CRUD actions are available Create/List/Update/Delete. All filters are based on Django's QuerySet, so List/Update/Delete has `--filter` (`-F`) argument for act as in Django's Lookups.

Add and filter actions use a `Form` issued from `ModelForm.get_form` to get default values, valid submitted data and return errors to user.

3.1 List

By default Admin CLI print the fields defined in `ModelAdmin.list_display`, in facts it is able to print:

- A field of model
- A callable that accepts one parameter for the model instance
- An attribute on the `ModelAdmin`
- An attribute on the model

You can specify which fields/attributes you want with `'--field'` (`'-f'`).

3.1.1 List model's instance

A basic listing is made as below:

```
$ ./manage.py cli user list
Username                      Email address                 First name                    Last name
zulu
admin
```

3.1.2 List specified fields

You can choose which field(s) you want to display with `'--field'` (`'-f'`):

```
$ ./manage.py cli user list -f id -f username
Id                    Username
1                     zulu
2                     admin
```

3.1.3 Filter specified fields

With Django's QuerySet syntax `'--filter'` (`'-F'`):

```
$ ./manage.py cli user list -F id=1
Username                      Email address                 First name                    Last name
zulu
```

3.1.4 Order by

And of course ordering with `'--order'` (`'-o'`):

```
$ ./manage.py cli user list -f username -o username
Username
admin
zulu
```

The reverse ordering is made by prefixing field's name by `'~'`, (instead of `'-'` like in Django):

```
$ ./manage.py cli user list -f username -o ~username
Username
zulu
admin
```

3.2 Add

Every field must be set with `'--field'` (`'-f'`). `ForeignKey` is defined by their primary key's value. Same for `ManyToManyField` except it's defined by with `','` as separator.

Note: This action uses a Django Form for validate data submited by user. If your `ModelAdmin` has `add_form` attribute (like `User` one), it will be used for data validation otherwise `ModelAdmin.get_form` is used.

3.2.1 Add an instance

Create an instance with `domain` and `name` as `CharField`:

```
$ ./manage.py cli site add -f domain=mysite.org -f 'name=My site'
Created 'mysite.org'
```

3.3 Update

Updates are made one by one, on every instances matching with given filters (`'--filter'`). It updates only field specified by `'--field'`.

3.3.1 Update an instance

Update an instance found from its `domain`:

```
$ ./manage.py cli site update -F domain=mysite.org -f 'name=New name'
Update 'mysite.org' ? [Yes|No|All|Cancel] y
Updated 'mysite.org'
```

3.4 Delete

Delete every instance matching with given filters (`'--filter'`).

3.4.1 Delete an instance

Delete an instance found from its `domain`:

```
$ ./manage.py cli site delete -F domain=mysite.org
Delete 'mysite.org' ? [Yes|No|All|Cancel] y
Deleted 'mysite.org'
```

3.5 Describe

Display `Model`'s field and `ModelAdmin.action`:

```
$ ./manage.py cli site describe
MODEL:
Name (Verbose)                 Type            Null  Blank Choices              Default         Help
id (ID)                        AutoField       0     1     []                   None
domain (domain name)           CharField       0     0     []
name (display name)            CharField       0     0     []
```

CHAPTER 4

Resctrict access to users

Put a `dict` named `ADMIN_CLI_USERS` in `settings.py`. It must have the following format:

```
ADMIN_CLI_USERS = {
  'login': 'RW',
}
```

Keys are UID or username, values are rights ‘R’ for read, ‘W’ for write/update/delete and ‘RW’ for both.

By default `ADMIN_CLI_USERS` is `{}` which allows all users to make all operations.

CHAPTER 5

Contributing

All project management tools are on GitHub:

- Bug tracking are in issues
- Patches are submitted as pull requests

5.1 Workflow

1. Fork project on GitHub.com
2. Make changes (and new unit tests if needed)
3. Ensure all tests are ok
4. Go to your fork and click on “Create pull request”

5.2 Tests

All tests are simply launched by:

```
python setup.py test
```

Or with coverage:

```
coverage run setup.py test
coverage html
```

5.3 Writing documentation

This documentation is built with Sphinx, make your local docs with following commands:

```
make html
cd _build/html
python -m SimpleHTTPServer
```

5.4 Online resources

- Code repository
- Documentation
- Travis CI server
- Coveralls report
- Landscape

CHAPTER 6

Indices and tables

- genindex
- modindex
- search