django-extensions Documentation

Release 1.7.4

Michael Trier, Bas van Oostveen, and contributors

September 15, 2016

Contents

Django Extensions Shell is the *shell_plus* command extracted from the excellent Django Extensions project into its own project. In most projects I've been only using the *shell_plus* command and to me it feels much better to only introduce a dependency on this specific code in projects then to depend on the whole Django Extensions project.

CHAPTER 1

Quickstart

You can get Django Extensions Shell by using pip or easy_install:

```
$ pip install django-extensions-shell
or
$ easy_install django-extensions-shell
```

If you want to install it from source, grab the git repository and run setup.py:

```
$ git clone git://github.com/shanx/django-extensions-shell.git
$ cd django-extensions-shell
$ python setup.py install
```

Now you will need to add the *django_extensions_shell* application to the INSTALLED_APPS setting of your Django project *settings.py* file.:

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

For more detailed instructions check out our `installation_instructions`. Enjoy.

CHAPTER 2

Compatibility with versions of Python and Django

This command will periodically follow updates done within Django Extensions so please check at: the Django Extensions page

CHAPTER 3

Contents

3.1 shell_plus

synopsis Django shell with autoloading of the apps database models

3.1.1 Interactive Python Shells

There is support for three different types of interactive python shells.

IPython:

```
$ ./manage.py shell_plus --ipython
```

BPython:

```
$ ./manage.py shell_plus --bpython
```

Python:

```
$ ./manage.py shell_plus --plain
```

The default resolution order is: bpython, ipython, python.

You can also set the configuration option SHELL_PLUS to explicitly specify which version you want.

```
# Always use IPython for shell_plus
SHELL_PLUS = "ipython"
```

It is also possible to use IPython Notebook, an interactive Python shell which uses a web browser as its user interface, as an alternative shell:

```
$ ./manage.py shell_plus --notebook
```

In addition to being savable, IPython Notebooks can be updated (while running) to reflect changes in a Django application's code with the menu command *Kernel* > *Restart*.

3.1.2 Configuration

Sometimes, models from your own apps and other people's apps have colliding names, or you may want to completely skip loading an app's models. Here are some examples of how to do that.

Note: These settings are only used inside shell_plus and will not affect your environment.

```
# Rename the automatic loaded module Messages in the app blog to blog_messages.
SHELL_PLUS_MODEL_ALIASES = {'blog': {'Messages': 'blog_messages'},}
}
```

```
# Prefix all automatically loaded models in the app blog with myblog.
SHELL_PLUS_APP_PREFIXES = {'blog': 'myblog',}
}
```

```
# Dont load the 'sites' app, and skip the model 'pictures' in the app 'blog'
SHELL_PLUS_DONT_LOAD = ['sites', 'blog.pictures']
}
```

You can also combine model_aliases and dont_load.

It is possible to ignore autoloaded modules when using manage.py, like:

```
$ ./manage.py shell_plus --dont-load app1 --dont-load app2.module1
```

Commandline parameters and settings in the configuration file are merged, so you can safely append modules to ignore from the commandline for one-time usage.

There are two settings that you can use to pass your custom options to the IPython Notebook in your Django settings.

The first one is `NOTEBOOK_ARGUMENTS` that can be used to hold those options that available via:

```
$ ipython notebook -h
```

For example:

```
NOTEBOOK_ARGUMENTS = [
    '--ip=x.x.x.x',
    '--port=xx',
]
```

Another one is `IPYTHON_ARGUMENTS` that for those options that available via:

```
$ ipython -h
```

The Django settings module and database models are auto-loaded into the interactive shell's global namespace also for IPython Notebook.

Auto-loading is done by a custom IPython extension which is activated by default by passing the `--ext django_extensions.management.notebook_extension` argument to the Notebook. If you need to pass custom options to the IPython Notebook, you can override the default options in your Django settings using the `IPYTHON_ARGUMENTS` setting. For example:

```
IPYTHON_ARGUMENTS = [
    '--ext', 'django_extensions.management.notebook_extension',
    '--ext', 'myproject.notebook_extension',
    '--debug',
]
```

To activate auto-loading, remember to either include the django-extensions' default notebook extension or copy its auto-loading code into your own extension.

Note that the IPython Notebook feature doesn't currently honor the `--dont-load` option.

3.1.3 Additional Imports

In addition to importing the models you can specify other items to import by default. These are specified in SHELL_PLUS_PRE_IMPORTS and SHELL_PLUS_POST_IMPORTS. The former is imported before any other imports (such as the default models import) and the latter is imported after any other imports. Both have similar syntax. So in your settings.py file:

```
SHELL_PLUS_PRE_IMPORTS = (
    ('module.submodule1', ('class1', 'function2')),
    ('module.submodule2', 'function3'),
    ('module.submodule3', '*'),
    'module.submodule4'
)
```

The above example would directly translate to the following python code which would be executed before the automatic imports:

```
from module.submodule1 import class1, function2
from module.submodule2 import function3
from module.submodule3 import *
import module.submodule4
```

These symbols will be available as soon as the shell starts.

3.1.4 Database application signature

If using PostgreSQL the `application_name` is set by default to `django_shell` to help identify queries made under shell_plus.

3.1.5 SQL queries

It is possible to print SQL queries as they're executed in shell_plus like:

```
$ ./manage.py shell_plus --print-sql
```

You can also set the configuration option SHELL_PLUS_PRINT_SQL to omit the above command line option.

```
# print SQL queries in shell_plus
SHELL_PLUS_PRINT_SQL = True
```