django-environments Documentation

Release django+environments-4869c39-py2.7

Goeie Jongens

February 15, 2015

Contents

django-environments helps you manage different settings within a Django project, and easily select those settings from the command line or from WSGI, all with "maximum DRY™".

On the command line, you specify your project and settings using environment variables and (mostly) *shell functions*. When you *run your application via WSGI*, a simple naming convention determines which *settings* to use based on the name of the WSGI file. All this helps to minimize the number of code changes and other file updates when working across different environments.

CHAPTER 1

Installation

django-environments is available as both a PyPI package and as "source" on Github.

1.1 Prerequisites

django-environments should run on any modern version of Python (v2.6 and up) and Django. It was tested to be compatible with Django 1.7, but should run on older versions as well.

Note: As django-environments make heavy use of Unix Shell scripts, it will only run on Unix- variants (Linux/Mac OS X). Windows support is currently not in scope.

1.1.1 Compatibility with virtualenv

Please note django-environments does not in any way depend on virtualenv, although it can be used together with virtualenv quite well.

When using django-environments within a single virtualenv environment, you can switch between Django projects as often as you like. If you use virtualenvwrapper, use `bin/postactivate` and `bin/predeactivate` for calling djenv and djexit respectively.

1.1.2 Compatibility with Python < 2.6

In the example settings files, `from .. import *` is used. You will need to change this to `from <project>.settings import *` for older versions of Python. The downside is that you will have to include the project name in your settings, which is a violation of the DRY principle that django-environments tries to live by.

1.2 Regular install from PyPI

See also:

Prerequisites

Installing django-environments into your existing Django project as a regular Python package is easy:

```
$ pip install django-environments
```

This installs two things:

1. A `djenv` Python package containing some default *settings modules* you can "extend" from.
2. A set of *Bash scripts and Shell functions* for easily switching between Django settings and projects.

For more information on using django-environments, see *Usage*.

1.3 Development install from Github

If you plan to make any changes to django-environments or want to be able to run the *example project*, you probably want to install django-environments directly from Github and set it up for local development:

```
$ cd ~/dev # or wherever
$ git clone https://github.com/yvandermeer/django-environments.git
$ cd django-environments
$ mkvirtualenv django-environments
$ pip install -r requirements/libs-dev.txt
```

This installs the bare requirements for local development and deploys django-environments itself using "easy_install develop".

Note: The above example assumes you use virtualenvwrapper installed. To use plain `virtualenv`:

```
$ virtualenv ~/.virtualenvs/django-environments
$ source ~/.virtualenvs/django-environments/bin/activate
```

From here, you can do a number of things:

- Run the *example project*.
- *Build the documentation locally*.
- Use your checkout of django-environments for *inclusion in other projects* under development.

1.4 Building the documentation locally

Documention is provided in Sphinx format in the `docs` subdirectory of the project. Assuming you have completed a *development installation*, you can build the HTML version of the documentation yourself:

```
$ cd docs
$ make html
```

Alternatively, you can browse the built documentation on Read the Docs.

1.5 Inclusion in other projects

If you want to (temporarily) use a locally checked out version of django-environments (as opposed to an official PyPI distribution) in other projects, you can do so easily using `pip install -e` (a.k.a. `easy_install develop`).

Note: The following example assumes you have succesfully completed a *development install* of django-environments into `~/dev/django-environments`.

First, activate the *virtualenv* for your other project:

```
$ workon some-other-project
```

Next, all you need to do is simply install django-environments as an "editable" pip requirement:

```
$ pip install -e ~/dev/django-environments
```

You should now be able to source the file containing the django-environments shell functions:

```
$ source djenvlib.sh
$ type djenv | head -n 1
djenv is a function
```

Also, the `djenv` package should now be available on the Python path:

```
$ python -c "import djenv; print djenv.__version__"
1.0 # or current version
```

Note: Any changes to the Python code in your original django-environments checkout should now be directly reflected in your other project. Changes to any shell script however, require that you re-run `pip install -e`.

For more information, see *Usage*.

See also:

Migration from the version on Bitbucket

CHAPTER 2

Usage

2.1 Getting started

django-environments is a very light-weight package that aims to make your life as Django developer easier. While it advocates certain patterns in setting up your Django project (e.g. an *"inheritance model" for your settings*), there are very few things that django-environments enforces on you.

To demonstrate, here is a quick-start guide to getting a first-time Django project up and running together with django-environments.

2.1.1 Starting a basic Django project

First, let's create a virtualenv with Django and django-environments installed and initialize an empty Django project:

```
$ mkvirtualenv djenv-example
$ pip install Django # installs latest version of Django
$ cd ~/dev/ && mkdir djenv-example # or wherever you like your project to live
$ djadmin startproject example djenv-example
```

This should now give you something like this:

```
djenv-example/
    example/
        __init__.py
        settings.py
        urls.py
        wsgi.py
    manage.py
```

2.1.2 Install django-environments

```
$ pip install django-environments
$ source djenvlib.sh
$ type djenv | head -n 1
djenv is a function # it works!
```

See also:

Installation

2.1.3 Enable django-environments in your Django project

To enable django-environments for your Django project, add one line to the top of your `settings.py`:

```
1 from djenv.settings.core import *
2 # ...
```

Note: This adds four custom settings that make further configuration easier. See `djenv.settings.core` for details.

Use the django-environments *setproject* command to set the `PROJECT_ROOT`:

```
$ setproject ~/dev/djenv-example/
Warning: no django projects found
$ echo $PROJECT_ROOT
/Users/yuri/dev/djenv-example
```

Note: The warning is because django-environments expects a `settings` *package* (containing an `__init__.py`) instead of a single `settings.py`. This is a known issue.

See also *Settings management*.

Now use the *djenv* command to tell django-environments the name of our `DJANGO_PROJECT`, and it will respond with some useful environment information:

```
$ djenv example
Welcome to djenv-example/example. Environment info:
PROJECT_ROOT: '/Users/yuri/dev/djenv-example'
DJANGO_PROJECT: 'example'
DJANGO_SETTINGS_MODULE: 'example.settings'
PYTHONPATH: '/Users/yuri/dev/djenv-example:/Users/yuri/dev/djenv-example/lib:'
```

That is pretty much it!

2.1.4 Useful commands

You can now use handy *commands* to navigate within your project, such as going to your Django project directory using *cdjango*:

```
$ cdjango && pwd
/Users/yuri/dev/djenv-example/example
```

or back to the project root using *cdroot*:

```
$ cdroot && pwd
/Users/yuri/dev/djenv-example
```

Note: It may not surprise you that these commands were inspired by some of the very useful commands that virtualenvrapper provides, such as `cdvirtualenv` and `cdsitepackages`.

If you quickly want know what the value of a Django setting is given the currently active django settings module, use the *get_django_setting* command:

```
$ get_django_setting ROOT_URLCONF
example.urls
```

See also:

A full list of *available commands*

2.1.5 Next steps

Now that you have the basics installed, you can further optimize and organize your Django settings. For example, the default Django 1.7 `settings.py` defines a `BASE_DIR`:

```
# ...
BASE_DIR = os.path.dirname(os.path.dirname(__file__))
# ...

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

But using django-environments, you can simply use `PROJECT_ROOT`:

```
from djenv.settings.core import * # sets PROJECT_ROOT
# ...

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(PROJECT_ROOT, 'db.sqlite3'),
    }
}
```

In this particular case though, you could even use `djenv.settings.database`:

```
from djenv.settings.core import *
from djenv.settings.database import * # defines DATABASES_DEFAULT
# ...

DATABASES['default'] = DATABASES_DEFAULT['sqlite']
```

You should also consider organizing your settings in a hierarchical structure – see *Settings management*.

2.2 Available commands

If everything works okay, the following shell functions are created:

- **djp (tab completion)** Start working on a project (first argument, optional second argument is the Django project, default "www"), automatically activating the virtualenv with the name of the project and selecting `settings.env.local` using *djenv*. Virtualenv and empty project are created if they do not yet exist.
- **setproject** Set the `PROJECT_ROOT` to either current or specified directory.
- **djenv (tab completion)** switch to different *settings* or another Django project.
- **cdroot (tab completion)** go to current `PROJECT_ROOT`.
- **cdlib (tab completion)** go to subdirectory 'lib' of the current `PROJECT_ROOT`.
- **cdjango (tab completion)** go to Django project root (one lower than project root).
- **djadmin (install tab completion yourself)** shorthand for django-admin.py, which you should use instead of manage.py (unless you want to tweak things).
- **runserver** perform `django-admin.py runserver <port>`, using `LOCAL_SERVER_PORT` if defined. Use option `-p` to bind to your network IP address.
- **djbrowse** points the browser to the server listening on `LOCAL_SERVER_PORT` in the current settings.
- **djvirtualbrowse** Points the browser to the named virtual host for the current settings. Assumes Apache is running as reverse proxy; see `bin/create_apache_vhost_conf` for more information.
- **pipup (tab completion)** call `pip install` with the appropriate file listing the project's requirements.
- **removeorphanpycs** remove .pyc files without a corresponding .py.
- **removeemptydirs** remove all empty directories in the project (calls *removeorphanpycs* first).
- **pycompile** compile all .py files - handy for web server environments, calls *removeorphanpycs* afterwards.
- **get_django_setting** get a value from the current settings module, useful for your own scripts (also see the experimental `import_django_settings`).
- **djexit** leave the current Django project.

See `bin/djenvlib.sh` for the more information.

2.3 Settings management

It is often necessary for a Django project to have different settings depending on the system environment, such as development, test, staging/acceptance and production. However, even on a single computer it can be helpful to have an easy way to switch between settings (e.g. to quickly simulate different environments).

django-environments helps you manage different Django settings within a Django project, and easily select settings from the command line or from WSGI, all with "maximum DRY™".

Contents

2.3.1 Definining your settings

Instead of using a single `settings.py` module, django-environments expects you to organize your settings in a Python *package*, for example:

```
mysite/
    settings/
        __init__.py
        env/
            __init__.py
            development.py
            production.py
        base.py
```

Additionally, it suggests (but does not dictate) you use an inheritance/generalization model, simply by "inheriting" from more generic settings using `from <package> import *`, and overruling the settings as needed:

```
# mysite/settings/base.py
from djenv.settings import *

INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'mysite',
)

# mysite/settings/env/development.py
from mysite.settings.base import *

DEBUG = True

INSTALLED_APPS += (
    'debug_toolbar',
)
```

2.3.2 Available settings modules

djenv.settings.core

The only *required* settings to include in order to use django environments.

This module defines four basic settings which are specific to django-environments:

djenv.settings.core.**PROJECT_ROOT = '/path/to/your/project'**
The full path to the root directory of your Django project.

This is useful for defining other settings such as STATIC_ROOT, and is used by the *cdroot* command.

djenv.settings.core.**PROJECT = 'project'**
The basename of the PROJECT_ROOT.

djenv.settings.core.**DJANGO_PROJECT = 'mysite'**
The basename of the directory containing your Django project's ROOT_URLCONF (urls.py).

djenv.settings.core.**DJANGO_PROJECT_DIR = '/path/to/your/project/mysite'**
The full path to the DJANGO_PROJECT directory.

djenv.settings.generic

Sets some basic "sane defaults" for various Django settings.

Settings defined here include the ROOT_URLCONF, STATIC_ROOT, STATIC_URL and basic MIDDLEWARE_CLASSES.

djenv.settings.generic.**LOCAL_SERVER_PORT = 8001**
The HTTP port used when running the Django development server using the *runserver* command.

djenv.settings.database

djenv.settings.log

djenv.settings.template

djenv.settings.cache

2.4 WSGI deployment

2.4.1 Using Apache mod_wsgi

Should you wish to use the settings in for instance settings/env/staging.py, simply copy the example mysite/deploy/development.wsgi to mysite/deploy/staging.wsgi, or make staging.wsgi a symlink (if your Apache configuration allows it, which is normally the case). Next, add something like this to your httpd.conf:

```
WSGIScriptAlias / /Users/spanky/repos/django-environments/mysite/deploy/staging.wsgi
```

And restart Apache. The identifier 'staging' in staging.wsgi will automatically make sure settings.env.staging is used. Create other .wsgi files for other environment settings.

Refer to the source of the provided WSGI script to see how specific directories, like a virtualenv site-packages directory, can be prepended to sys.path, overruling standard Python environment settings.

2.4.2 Automatic generation of local WSGI links and settings file

If you want your WSGI setup done as quickly as possible, activate an environment - either directly via your bin/initenv or through virtualenv - and execute bin/setup_local_wsgi.sh <environment>, e.g.:

```
$ bin/setup_local_wsgi.sh staging
```

This will create a `deploy/local.wsgi` symbolic link to staging.wsgi and will create a `settings/env/local.py` with default contents for a given environment. Now, you only need to update `settings.env.local` with those settings you want to keep absolutely local, like those containing user ids and passwords. Keep in mind the script will overwrite existing `local.py` settings files!

2.5 Running the example project

To demonstrate (and test) django-environments, it comes with an example Django project, located in the top-level `example` directory in the source code.

Note: When *installing* django-environments from PyPI, only the bare essentials are installed. To run you the example project, follow the instructions for a *local development setup*.

Everything below assumes you already have a local checkout from the Github repository.

To run the example project, you can use the `example/bin/initenv.sh` example script:

```
$ workon django-environments
$ source ~/Development/Projects/django-environments/example/bin/initenv.sh
$ runserver
```

To make it even easier, you may want to hook into virtualenvwrapper's *postactivate* hook:

```
$ echo 'source ~/dev/django-environments/example/bin/initenv.sh' >> $WORKON_HOME/django-environments/
$ workon django-environments
$ runserver
```

For more details on, see the contents of the `example/bin/initenv.sh` file.

> **Warning:** Alternatively, instead of sourcing the example `initenv.sh` script from the `postactivate` script, you might be tempted to symlink the file. This won't work however, as it breaks the current `PROJECT_ROOT` detection in the example `initenv.sh` script:
>
> ```
> PROJECT_ROOT=$(cd -P "$(dirname "${BASH_SOURCE[0]}")"/.. && pwd)
> ```

2.5.1 Directories

- The `mysite/example/settings` directory replaces `settings.py` and contains the default settings in `generic.py`, whose contents are imported in `__init__.py`.
- The `mysite/settings/env` directory contains the different settings files for every environment.
- All .wsgi files in the `mysite/deploy` folder are normally equal, except for the `sys.path` configuration. Their respective filenames are used to determine which settings to import. If your Apache configuration allows it, you could use symlinks instead of copies.

2.5.2 Remarks

Todo

Check if this information is still accurate

- `urls.py` is just there to demonstrate the `SERVE_MEDIA` setting, which is not essential anyway.
- `manage.py` was removed as the generated default ignores `$DJANGO_SETTINGS_MODULE`, simply importing 'settings' instead.
- the Django `startapp` command will create new apps in `$DJANGO_PROJECT/settings/env`. Apparently, Django uses the basename of the settings `__file__` as a reference point for the new app.

2.6 Migration from the version on Bitbucket

Are you migrating from the version of django-environments on Bitbucket? You should be able to get everything working with a few minor (but important) modifications.

2.6.1 Remove django-environments from "hg externals"

Before, it was common to include django-environments as an external Mercurial repository in your project using a `requirements/externals-prd.txt` file, which would contain something like this:

```
https://bitbucket.org/goeiejongens/django-environments#bed884e334c2
```

If you wish to migrate to this Github "fork", you have two options:

- *Regular install from PyPI*
- *Development install from Github*

2.6.2 Reverse use of `.sh` for shell scripts

The use of the `.sh` extension for the shell scripts has been reversed: scripts that are meant to be directly executed now *do not* have an extension, and scripts that are meant to be "sourced" *do* have an `.sh` extension. If you call these scripts in any of your own scripts, you should update accordingly.

2.6.3 Top-level "etc" symlink no longer necessary

Because django-environments is now setuptools-compliant, it will install itself into the Python "site-packages" directory (both if you do a *standard PyPI install* or an *"editable" install from Github*), and will therefore be automatically available on your Python path.

However, see next item.

2.6.4 Python "etc" package renamed to "djenv"

To avoid naming conflicts with other Python packages, `djenv` seemed like a better package name. You should probably do a global search/replace for:

- `etc.settings` => `djenv.settings`
- `etc.urls` => `djenv.urls`

2.6.5 g0j0 settings proxy removed

The `template settings` previously included the "g0j0.context_processors.settings_proxy" in the default list of `TEMPLATE_CONTEXT_PROCESSORS`. This `g0j0` dependency has now been removed to keep django-environments generic, so please add it manually in your project settings if you require it.

2.6.6 About your global django-environments installation

There's a good chance that you also have a global installation of django-environments (e.g. to activate projects using the *djp* command). If so, you probably currently source the Bash scripts containing the various *shell functions* from your `.bashrc` like so:

```
source $REPOS/django-environments/bin/djenvlib
source $REPOS/django-environments/bin/djenv.mercurial # optionally
```

If you want, you can simply keep working this way (replacing of course the checkout from Bitbucket with a *clone from Github*. Having a central checkout of django-environments also allows you to *contribute* to django-environments itself *while using it in other projects*.

However, if you simply want to have a global installation of django-environments, you could also simply install it from PyPI directly into your global site packages:

```
$ deactivate # make sure you are not in any activated virtualenv
$ PIP_REQUIRE_VIRTUALENV=false pip install django-environments
```

This installs the shell scripts into a common `bin` directory (the exact location depends on your Python installation):

```
$ which djenvlib.sh
/usr/local/bin/djenvlib.sh # Location for Homebrew Python on Mac OS X
```

You should then be able to simply source the scripts from your `.bashrc` like so:

```
source djenvlib.sh
source djenv.mercurial.sh
```

CHAPTER 3

Release notes

3.1 dev

- Added *Getting started* section to documentation.
- Fixed some flake8 errors/warnings

3.2 1.0a5

- Fixed critical error where non-wheel installation from the PyPI package would fail due to the "README.rst" not being included in the distribution package.

3.3 1.0a4

- First release to PyPI. See *Migration from the version on Bitbucket*
- Renamed top-level `etc` package to `djenv`
- Reversed use of `.sh` for shell scripts: executable scripts no longer have the `.sh` extension, whereas shell scripts meant to be `source`'d now *do* have the extension
- Added Django 1.7 compatibility
- Moved "mysite" example project (and related files/directories) to top-level "example" directory
- Expanded documentation, now implemented as Sphinx docs and published on Read the Docs.

CHAPTER 4

Credits & contributing

The django-environments project was originally started by Goeie Jongens and most of its code was written by Vincent Hillenbrink.

Minor contributions have been made by Yuri van der Meer, including its release to PyPI.

Note: django-environments was originally created as a private Mercurial repository and later on it started being pushed to Bitbucket periodically. That version is meant to be included into your project as an "external" Mercurial repository – it does not use Setuptools/distutils.

In February 2015, it was migrated to Github (using git-remote-hg) and packaged using Setuptools for distribution through PyPI.

The version as it exists on Github should be considered a fork of the original version on Bitbucket.

If you want to contribute to django-environments, feel free to fork the project on Github.

d

D

L

P