django_dbdev Documentation

Release 1.2.1

Espen Angell Kristiansen

Mar 07, 2018

Contents

django_dbdev is a set of Django management commands that makes it very easy work with database servers during development.

We provide Django management commands to:

- Setup a clean and isolated database environment in a temporary folder. This means that you do not have to touch ANY globally installed database configs or databases.
- Create and destroy this isolated database enviroment, including all those hard to remember commands to create the database, create a user with the correct privileges etc.
- Load database dumps.
- Backup and restore your database.

CHAPTER 1

Supported databases

- PostgreSQL
- Sqlite3
- MySQL
- MariaDB (same backend as MySQL)

You can also easily *add support for more databases*.

CHAPTER 2

Help

2.1 Getting started

2.1.1 Install

```
$ pip install django_dbdev
```

2.1.2 Configure your Django project

Add `django_dbdev` to `INSTALLED_APPS`, and setup one of the database backends (below).

Setup for PostgreSQL

Add the following to your Django settings:

```
from django_dbdev.backends.postgres import DBSETTINGS

DATABASES = {
    'default': DBSETTINGS
}
```

Setup for sqlite3

Add the following to your Django settings:

```
from django_dbdev.backends.sqlite import DBSETTINGS

DATABASES = {
    'default': DBSETTINGS
}
```

Setup for MySQL

Add the following to your Django settings:

```
from django_dbdev.backends.mysql import DBSETTINGS

DATABASES = {
    'default': DBSETTINGS
}
```

Note: If you use the mariadb or mysql packages for homebrew on OSX, you must also set *DBDEV_MYSQL_BASEDIR* as an environent variable or Django setting. We recommend using an environment variable to avoid affecting other developers on the same project, and because it allows you to fix it for all your Django projects.

For mariadb it will look something like this:

```
export DBDEV_MYSQL_BASEDIR=/usr/local/Cellar/mariadb/10.0.10/
```

The version number will vary. For mysql the only difference will be the name of the directory (mysql instead of mariadb).

2.1.3 Avoiding port-number-crashes

If you are developing multiple projects simultaneously, or just have a lot of stuff running on various ports and want to avoid a crash, you can specify the port for dbdev by adding this line below `DATABASES` in `settings.py`:

```
DATABASES['default']['PORT'] = <my_random_port_number>
```

where *my_random_port_number* is the port you want your dbdev-database to run on. We recommend using a port somewhere in the range 20.000-50.000.

2.1.4 Developing for multiple database backends

The `dbdev_testproject` explained in *Develop django_dbdev* in an example of one such setup. The only thing required is a way of specifying which DB backend to use, and that can be done in many ways.

2.2 Working with django_dbdev

2.2.1 Create your database

To setup an isolated database environment with a development database and user in the `dbdev_tempdata/<dbengine>/` directory, run:

```
$ python manage.py dbdev_init
```

After running this, you should be able to initialize the database using `syncdb`:

```
$ python manage.py syncdb
```

Note: If you do not want to use `dbdev_tempdata/` as the data directory, simply set the `DBDEV_DATADIR`-setting (in your settings.py) to something else.

Note: If `dbdev_init` fails, read the output to determine what went wrong. Then run `debdev_destroy` (explained below) to remove everything that `dbdev_init` created before you try again.

2.2.2 Destroy or re-init the database environment

You can destroy your development database and all of the files associated with it except for backups using:

```
$ python manage.py dbdev_destroy
```

We also provide a shortcut that is equivalent to running `dbdev_destroy` followed by `dbdev_init`:

```
$ python manage.py dbdev_reinit
```

2.2.3 Load a database dump

You can load a database dump using:

```
$ python manage.py dbdev_loaddump /path/to/dumpfile.sql
```

2.2.4 Backup and restore

You can dump your current database into `dbdev_tempdata/<backend>-backups/` with:

```
$ python manage.py dbdev_backup
```

And you can restore the last backup using:

```
$ python manage.py dbdev_restore
```

Warning: When you restore a backup, `dbdev_restore` runs `dbdev_reinit` to ensure you restore to a clean database enviroment. This means that you will loose all data on the server when you restore a backup, uncluding any extra database you may have added.

You can list all backups with:

```
$ python manage.py dbdev_listbackups
```

And restore a backup by name with:

```
$ python manage.py dbdev_restore <path-to-backupdir>
```

If you make a more important backup, you can use a name for it:

```
$ python manage.py dbdev_backup -n mybackup
... and restore with
$ python manage.py dbdev_restore -n mybackup
```

Backup names are unique, so you will not be able to create multiple backups with the same name.

Finally, you can clear all backups for your backend using:

```
$ python manage.py dbdev_clearbackups
```

Note: `dbdev_destroy` and `dbdev_reinit` does not affect backups.

2.2.5 Start your database shell

Logging into your database shell is provided by the standard Django management command `dbshell`:

```
$ python manage.py dbshell
```

2.2.6 Help remembering all those database debugging comands and quirks

Each backend provides a guide with tips and examples. Run:

```
$ python manage.py dbdev_guide
```

To show this guide.

2.3 Settings

django_dbdev is configured using Django settings. The following settings is available.

2.3.1 Common for all backends

`django.conf.settings.`**`DBDEV_DATADIR`**
: The directory where we store the data for django_dbdev. Each backend creates its own subdirectory where they store settings, database-data etc. We also store backups in subdirectories of this directory.

  Defaults to `dbdev_tempdata`.

2.3.2 Posgres backend settings

`django.conf.settings.`**`DBDEV_POSTGRES_EXECUTABLE`**
: Path to the `postgres` executable. Defaults to `postgres`.

`django.conf.settings.`**`DBDEV_POSTGRES_PG_CTL_EXECUTABLE`**
: Path to the `pg_ctl` executable. Defaults to `pg_ctl`.

`django.conf.settings.`**`DBDEV_POSTGRES_PSQL_EXECUTABLE`**
: Path to the `psql` executable. Defaults to `psql`.

`django.conf.settings.`**`DBDEV_POSTGRES_CREATEDB_EXECUTABLE`**
Path to the `createdb` executable. Defaults to `createdb`.

`django.conf.settings.`**`DBDEV_POSTGRES_PG_DUMP_EXECUTABLE`**
Path to the `pg_dump` executable. Defaults to `pg_dump`.

2.3.3 MySQL backend settings

`django.conf.settings.`**`DBDEV_MYSQL_INSTALL_DB_EXECUTABLE`**
Path to the `mysql_install_db` executable. Defaults to `mysql_install_db`.

`django.conf.settings.`**`DBDEV_MYSQLD_EXECUTABLE`**
Path to the `mysqld_safe` executable. Defaults to `mysqld_safe`.

`django.conf.settings.`**`DBDEV_MYSQLADMIN_EXECUTABLE`**
Path to the `mysqladmin` executable. Defaults to `mysqladmin`.

`django.conf.settings.`**`DBDEV_MYSQL_EXECUTABLE`**
Path to the `mysql` executable. Defaults to `mysql`.

`django.conf.settings.`**`DBDEV_MYSQLDUMP_EXECUTABLE`**
Path to the `mysqldump` executable. Defaults to `mysqldump`.

`django.conf.settings.`**`DBDEV_MYSQL_BASEDIR`**
The path to the mysql *basedir* (where mysql default data is installed). You may have to set this if your MySQL/MariaDB is not configured correctly.

Can also be set as an environment variable, which is probably a better choice for projects with more than one developer.

Defaults to `None`.

2.4 Custom backend

Implementing a backend requires you to extend *`BaseDbdevBackend`* and override/implement the following methods:

- *`BaseDbdevBackend.init()`*
- *`BaseDbdevBackend.destroy()`*
- *`BaseDbdevBackend.run_database_server_in_foreground()`*
- *`BaseDbdevBackend.start_database_server()`*
- *`BaseDbdevBackend.stop_database_server()`*
- *`BaseDbdevBackend.backup()`*
- *`BaseDbdevBackend.restore()`*
- *`BaseDbdevBackend.serverinfo()`*

You must also:

- Create a template that the *`BaseDbdevBackend.guide()`* method can use.
- Add a DBSETTINGS dict to the module containing your backend (see the other backends).

2.4.1 Register your backend

You must register your backend some place that is always executed when Django starts up. The most natural place is in your development `settings.py`. Lets say you have implemented an Oracle backend, you need to tell django_dbdev to use your dbdev backend for the `django.db.backends.oracle` engine like this:

```
from django_dbdev import backendregistry

from mypackage.dbdev_backends.oracle import OracleBackend

backendregistry.register('django.db.backends.oracle', OracleBackend)
```

Note: You can replace the built in backends this way too. The registry is a dict mapping engine string to backend class, so registering a custom backend for an engine with a built in backend (like `django.db.backends.mysql`) does not raise any errors.

2.4.2 BaseDbdevBackend docs

class django_dbdev.backends.base.**BaseDbdevBackend**(*command*)

Bases: `future.types.newobject.newobject`

Abstract base class for dbdev backends.

Parameters **command** – A Django management command class. Will always be a subclass of `django_dbdev.management.commands._base.BaseDbdevCommand`.

init()

Create the database and grant the required previleges to the database user.

destroy()

Remove all the files for the database.

Typically stops any running database server and deletes the data directory.

run_database_server_in_foreground()

Run database server in the foreground.

start_database_server()

Start database server in the background.

stop_database_server()

Stop database server started with *start_database_server()*.

backup(*directory*)

Create a backup of the database.

Parameters **directory** – The backup directory to create the backup in.

restore(*directory*)

Restore a backup created with `backup.()`

Parameters **directory** – The backup directory to restore.

serverinfo()

Print information about the server.

Must at least tell if the server is running or not.

guide()
Print useful database specific commands and tips for the user.

Examples should include all the needed login info.

The idea is to avoid having to lookup those commonly needed database-specific management and connection commands that is needed from time to time.

The default expects the backend to create a Django template named `django_dbdev/<backend-class-name-lowercased>.rst`. The template gets the backend class as `backend` context variable, and the `dbsettings` context variable contains `<backendmodule>.DBSETTINGS`.

Use the ReStructuredText format for the text.

datadir
Get the path to the temporary data directory for this database backend.

The directory is created if it does not exist.

create_datadir_if_not_exists()
Create `datadir()` if it does not exist.

remove_datadir()
Remove the `datadir()`.

stdout
Shortcut for `self.command.stdout`.

Use for normal messages. I.E.:

```
self.stdout.write('Something useful here!')
```

stderr
Shortcut for `self.command.stderr`.

Use for error messages. I.E.:

```
self.stderr.write('An error of some sort')
```

root_backupdir
Get the path to the backup directory for this database backend.

create_timestamped_backupdir(*name=None*)
Create a timestamped directory within `root_backupdir()`.

Returns The path to the created directory.

reinit()
Destroy and re-initialize.

2.5 Develop django_dbdev

2.5.1 Install the requirements

Install the following:

1. Python:)
2. PIP
3. VirtualEnv

4. virtualenvwrapper
5. MySQL (you will need the development headers to build the Python module)
6. PostgreSQL (you will need the development headers to build the Python module)

2.5.2 Install required Python modules in a virtualenv

Create a virtualenv:

```
$ mkvirtualenv django_dbdev
```

Install the development requirements:

```
$ cd djangoproject/
$ pip install -r requirements_development.txt
```

2.5.3 Run the test project

Navigate to the testproject:

```
$ cd dbdev_testproject/
```

Test with postgres using:

```
$ DJANGO_SETTINGS_MODULE=dbdev_testproject.develop.settings.postgres python manage.py
```

Test with sqlite using:

```
$ DJANGO_SETTINGS_MODULE=dbdev_testproject.develop.settings.sqlite python manage.py
```

Test with mysql using:

```
$ DJANGO_SETTINGS_MODULE=dbdev_testproject.develop.settings.mysql python manage.py
```

2.5.4 Submit a patch

1. Fork the GitHub repository.
2. If you are making a major change, you should create an issue where you explain the change and the motivation. This will make it far less likely that the patch will be rejected.
3. When the patch is ready, send a pull request to `espenak`.

2.5.5 Release a new version

- Update `django_dbdev/version.json`
- Git commit the release with message `Release <version>`.
- Git tag the release with `<version>`.
- `python setup.py sdist`
- `twine upload dist/django-dbdev-<version>.tar.gz` (you may need to do `pip install twine`)

CHAPTER 3

Source and issues

For sources and issue tracker, see the GitHub page for the project. If you plan to submit a patch, please read *Develop django_dbdev*.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index