Django DevFixtures

Release 0.1.3

March 23, 2016

Contents

CHAPTER 1

Overview

docs	
tests	
package	

Share development fixtures across your team, with git commit id tracing and autodetect.

- Free software: BSD license

1.1 Installation

Currently this package requires git, psql, pg_dump createdb, dropdb and unzip to function.

```
pip install django-devfixtures
```

1.2 Configuration

Add **devfixtures** to **INSTALLED_APPS**.

```
settings.DEVFIXTURE_DIR         # path to directory where auto generated fixtures should be stored
settings.DEVFIXTURE_BACKUP_DIR  # path to where backups are stored when running restore
```

1.3 How it works

When you **create** a fixture (without any arguments) the management command will zip MEDIA_FILES and database dump to a file with naming <AUTHOR_DATE>+<COMMITID>+<CREATED_DATE>+<CREATOR>.zip.

The auto restore function will check from HEAD and backards in the commit tree, and when it finds a fixture file with that commit id, it will restore that version after a backup of the current state has been created. If the restore for some reason fails, it will attempt to restore the backuped fixture.

This works with the following criterias:

- You will not rebase/rewrite history, as commit ids might no longer match
- You will not delete migrations manually

It also have the limitations to unly work with PostgreSQL. And there are some current limitations, due to the fact that the implementation uses pg_dump etc for creating dumps. Requirements:

- The database name is defined in **settings.DATABASES['default']['NAME']**
- The running user have permissions to dropdb/createdb

1.4 Storing the dev_fixtures in git

Devfixtures can become large, if you have big set of media files. If you use github, you are encouraged to use git-lfs to store the files. Read about git lfs here: https://git-lfs.github.com/

Add devfixtures/* to your tracked git lfs files before you add your first fixture to git.

```
# git lfs track 'dev_fixtures/*'
```

1.5 Usage

Create fixture:

```
# ./manage.py devfixture create
```

Restore fixture:

```
# ./manage.py devfixture restore
```

Create manual fixture sharing, for example if you have a branch and you want some other developer to take a look at a bug. Run this and send the zip file to the other developer:

```
# ./manage.py devfixture create -f ~/Desktop/devfixture-for-peter-debugging.zip
```

To load a shared fixture:

```
# ./manage.py devfixture restore -f ~/Desktop/devfixture-for-peter-debugging.zip
```

When running restore, a backup is always created. You can restore it the same way as above.

```
# ./manage.py devfixture -h
```

1.6 Documentation

https://django-devfixtures.readthedocs.org/

1.7 Development

To run the all tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER 2

Reference

2.1 devfixtures

CHAPTER 3

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

3.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

3.2 Documentation improvements

Django DevFixtures could always use more documentation, whether as part of the official Django DevFixtures docs, in docstrings, or even on the web in blog posts, articles, and such.

3.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/dolphinkiss/django-devfixtures/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

3.4 Development

To set up *django-devfixtures* for local development:

1. Fork django-devfixtures (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/django-devfixtures.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

3.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`) [1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

3.4.2 Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 4

Authors

- Peter Lauri - https://github.com/peterlauri

CHAPTER 5

Changelog

5.1 0.1.3 (2016-02-02)

- If pg_dump fails, CommandError is raised.
- Updated documentation that recommends git lfs if dev fixtures will be stored in your github repo.

5.2 0.1.1 (2016-02-02)

- First release on PyPI.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

d

D