Protein Data Bank Parameters

Release 0.0.0

Mar 29, 2020

Contents

CHAPTER 1

Protein Data Bank Parameters

A Python package hosting the static parameters for the Protein Data Bank file formats.

1.1 Motivation

Handling Protein Data Bank data through Python requires a constantly retyping of the PDB format static parameters, such as, line parsing slices, atoms names, residue names, etc. This package hosts all those static parameters required to handle `.pdb` files.

1.2 Installation

```
`bash pip install --upgrade pdbparams `
```

CHAPTER 2

Usage

Protein Data Bank Parameters library is organized in different thematic modules.

For example, to access the `slice` objects required to slice `PDB` ATOM lines:

CHAPTER 3

Contributing

3.1 Fork this repository

Fork this repository before contributing. It is a better practice, possibly even enforced, that only Pull Request from forks are accepted. In my opinion this creates a cleaner representation of the whole contributions to the project.

3.2 Install for developers

First, clone the repository as described in the *section above*.

Create a dedicated Python environment where to develop the project.

If you are using `pip` follow the official instructions on Installing packages using pip and virtual environments, most likely what you want is:

```
python3 -m venv pdbparams
source pyprojskel/bin/activate
```

If you are using Anaconda go for:

```
conda create --name pyprojskel python=3.7
conda activate pdbparams
```

Where `pdbparams` is the name you wish to give to the environment dedicated to this project.

Either under *pip* or *conda*, install the package in `develop` mode, and also *tox*.

```
python setup.py develop
# for pip
pip install tox
# for conda
conda install tox -c conda-forge
```

Under this configuration the source you edit in the repository git folder is automatically reflected in the development installation.

Continue your implementation following the development guidelines described bellow.

3.3 Branch workflow

The following applies to external contributors, yet main developers can also follow these guidelines.

Branch workflow for development and contribution should follow the Gitflow Workflow guidelines. Please read careful through that guide. Here we highlight the general approach with some tasteful additions such as the `--no-ff` flag.

3.3.1 Clone your fork

Indeed the first thing to do is to clone your fork, and keep it up to date with the upstream:

```
git clone https://github.com/YOUR-USERNAME/Protein_Data_Bank_Parameters.git
cd into/cloned/fork-repo
git remote add upstream git://github.com/joaomcteixeira/Protein_Data_Bank_Parameters.
↪git
git fetch upstream
git checkout latest
git pull upstream latest
git push origin latest  # to send updates to your forked repository
```

3.3.2 New feature

To work on a new feature, branch out from the `latest` branch:

```
git checkout latest
git checkout -b feature_branch
```

Develop the feature and keep regular pushes to your fork with comprehensible commit messages.

3.3.3 Push to latest

To see your development accepted in the main project, you should create a Pull Request to the `latest` branch following the PULLREQUEST.rst guidelines.

Before submitting a Pull Request, verify your development branch passes all tests as *described bellow* . If you are developing new code you should also implement new test cases.

3.4 Uniformed Tests

Thanks to Tox we can have a uniform testing platform where all developers are forced to follow the same rules and, above all, all tests occur in a controlled Python environment.

With *Tox*, the testing setup can be defined in a configuration file, the tox.ini, which contains all the operations that are performed during the test phase. Therefore, to run the unified test suite, developers just need to execute `tox`, provided tox is installed in the Python environment in use.

```
pip install tox
# or
conda install tox -c conda-forge
```

Before creating a Pull Request from your branch, certify that all the tests pass correctly by running:

```
tox
```

These are exactly the same tests that will be performed in Travis-CI.

Also, you can run individual environments if you wish to test only specific functionalities, for example:

```
tox -e check  # code style and file compatibility
tox -e docs  # only builds the documentation
tox -e py37  # runs tests for python=3.7
```

CHAPTER 4

Source documentation

4.1 slices

PDB line slices objects.

CHAPTER 5

Changelog

CHAPTER 6

Authors

- Joao M. C. Teixeira (webpage, github)

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

P