vasp$_{t}ools Documentation$

Release 0.1.0

Zaid Hassan

Sep 13, 2021

Contents

Contents:

CHAPTER 1

VASP Tools

VASP Tools is a set of modules and scripts that automate routine tasks involving VASP files using a very intuitive CLI. The `/scripts` directory contains the scripts that implement the `/vasp` module to perform routine tasks on VASP files. This project is still a WIP and new scripts/modules will be added regularly over the next few weeks.

1.1 Requirements

As of now, this package is only supported on `Python>=3.5`. Since support for `Python 2.7` is set to be pulled by 2020, updates in the near future extending support to `Python<=3.0` seems unlikely.

The following libraries are required to run all the scripts and modules.

- numpy
- argparse
- sympy
- ujson
- jsonschema
- tabulate

For a full list of requirements, read requirements.txt. If not already present within the environment, they'll be installed as dependencies during setup.

1.2 Installation

The installation process is quite simple, ensure you have a working version of `Python>=3.5` installed and type the following into the console,

```
pip install vasp-tools
```

Any required libraries that aren't installed in the current environment will be automatically installed. This will also automatically install the scripts and add them to $PATH for easy access.

1.3 Compatibility

The package, so far, was only tested within a Linux environment and isn't officially compatible with Windows yet. The scripts can be compiled into executables using PyInstaller to work independently of python on any system, though it should be run in an environment with an identical OS. Use of VMs/Containers is suggested, though not tested as of yet.

1.4 Usage

The code present in `/vasp` can be imported in the form of standard modules. However, the primary purpose of this project was the creation of scripts (present in `/scripts`) to automate daily tasks faced by the Computational Chemist/Material Scientist. With this in mind, the scripts were designed to be extremely modular and user-friendly by implementing a dplyr -esque piping paradigm. For example, the process of:

1. Importing a molecule from a `POSCAR` file.
2. Rotating it into a certain configuration (90 degrees wrt the x-axis)
3. Positioning it at a specified point above a crystal taken from another `POSCAR` file
4. Fixing atomic positions within the crystal below a certain cutoff height
5. Converting the coordinates to `Direct` from `Cartesian` or vice versa
6. Save to a new `POSCAR` file.

can be implemented in a single line like so.

```
cat POSCAR1 | ./rotate.py -x 90 -y 10 | ./place-at.py "POSCAR2" 0.5 0.5 2.0 | ./fix-
↪upto.py 10.0 | ./cart-direct > POSCARnew
```

Alternatively, you can also call each script individually or pass "POSCAR1" as one of the positional arguments. For example,

```
./place-at.py "POSCAR2" "POSCAR1" 0.5 0.5 2.0
```

is perfectly equivalent to

```
cat POSCAR1 | ./place-at.py "POSCAR2" 0.5 0.5 2.0
```

Detailed instructions on how to use the scripts are available in docs.

Written with StackEdit

CHAPTER 2

Installation

2.1 Stable release

To install vasp_tools, run this command in your terminal:

```
$ pip install vasp_tools
```

This is the preferred method to install vasp_tools, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for vasp_tools can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/RexGalilae/vasp_tools
```

Or download the tarball:

```
$ curl  -OL https://github.com/RexGalilae/vasp_tools/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use vasp_tools in a project:

```
import vasp_tools
```

CHAPTER 4

Contributing

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

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/RexGalilae/vasp_tools/issues.

If you are 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.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/RexGalilae/vasp_tools/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 contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *vasp_tools* for local development.

1. Fork the *vasp_tools* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/vasp_tools.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv vasp_tools
$ cd vasp_tools/
$ python setup.py develop
```

4. Create a branch for local development:

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

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 vasp_tools tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. 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
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/RexGalilae/vasp_tools/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_vasp_tools
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search