paved Documentation

Release 0.2

David Eyk

November 08, 2016

Contents

Paved is a set of common tasks and helpers for quickly setting up a Paver-based project.

Contents:

CHAPTER 1

Paved

Paved provides common tasks for Paver-based projects.

1.1 Quick Start

To start using `paved`, inside your `pavement.py`:

```
from paved import *
```

This will set up the `options.paved` namespace, and give you one extremely useful task: `clean`, which takes care of cleaning up common clutter files like `*.pyc`, `*.pyo`, and `*~`. Of course, it's customizable using `options.paved.clean.patterns` and `options.paved.clean.dirs`.

1.2 Other Modules

Use the same pattern of `from paved.<module> import *` for other available modules:

- `paved.dist`: distribution-related tasks and shortcuts
- `paved.pkg`: some packaging-related tasks
- `paved.django`: Django-related tasks
- `paved.util`: some useful utility functions
- `paved.util.pycheck`: python code checking functions

1.3 Contributing

I'd love to have help with Paved. If you have common tasks that you're always copying and pasting from one pavement.py to the next, they probably have a place here.

The best way to contribute to Paved is to fork the project on Github and send pull requests to `eykd`.

CHAPTER 2

Quickstart

Paved comes with several task bundles configured to work out of the box for the most common use-cases. You use them by importing their members into your pavement.py's module namespace.

As a quick example, let's start a new Django project. We create our pavement.py:

```
"""pavement.py -- paver tasks for our new Django project!
"""
from paver.easy import *

__path__ = path(__file__).abspath().dirname()
```

Now we add these two lines:

```
from paved.django import manage

options.paved.django.manage_py = __path__ / 'myproject' / 'manage.py'
```

Suddenly, we can start doing things like:

```
$ paver manage syncdb
```

This will run our django project's *manage.py* file with *syncdb* as the first argument. All of Django's manage commands are available to us now.

Once we're ready for our first release, having made sure that our setup() metadata is all there, we add this to the imports:

```
from paved.dist import *
```

Now *sdist* is hooked up as described in Paver's documentation:

```
$ paver sdist
```

This produces a *bootstrap.py*, *setup.py*, *paver-minilib.zip*, as well as a (configurable) *MANIFEST.in* that does the right stuff by default. Finally, you'll find your source distribution file in the *dist/* folder, as you might expect.

Read the API documentation to find out more about the tasks and utilities available.

CHAPTER 3

API

Using Paved is pretty easy: just import what you need. The tasks will register themselves with Paver, and away you go! The one exception is with paved.util, which provides only helper functions and no tasks.

3.1 paved

paved – common paver tasks.

paved.paved.**clean**(*options*, *info*)
Clean up extra files littering the source tree.

options.paved.clean.dirs: directories to search recursively options.paved.clean.patterns: patterns to search for and remove

paved.paved.**printoptions**()
print paver options.

Prettified by json. *long_description* is removed

3.2 paved.pkg

paved.pkg – packaging tools for paved.

paved.pkg.**pip_install**(*args*)
Send the given arguments to *pip install*.

paved.pkg.**easy_install**(*args*)
Send the given arguments to *easy_install*.

3.3 paved.dist

paved.dist – distribution tasks

paved.dist.**sdist**()
Overrides sdist to make sure that our setup.py is generated.

paved.dist.**upload**()
Upload the package to PyPI.

paved.dist.**manifest**()
Guarantee the existence of a basic MANIFEST.in.

manifest doc: http://docs.python.org/distutils/sourcedist.html#manifest

options.paved.dist.manifest.include: set of files (or globs) to include with the *include* directive.

options.paved.dist.manifest.recursive_include: set of files (or globs) to include with the *recursive-include* directive.

options.paved.dist.manifest.prune: set of files (or globs) to exclude with the *prune* directive.

options.paved.dist.manifest.include_sphinx_docroot: True -> sphinx docroot is added as *graft*

options.paved.dist.manifest.include_sphinx_docroot: True -> sphinx builddir is added as *prune*

3.4 paved.docs

paved.sphinx – helpers and tasks for Sphinx documentation.

paved.docs.**sphinx_make**(**targets*)
Call the Sphinx Makefile with the specified targets.

options.paved.docs.path: the path to the Sphinx folder (where the Makefile resides).

paved.docs.**docs**()
Make Sphinx docs.

options.paved.docs.path: the path to the Sphinx folder (where the Makefile resides).

options.paved.docs.targets: the Make targets to send to *sphinx_make*. Default is *html*.

paved.docs.**clean_docs**()
Clean Sphinx docs.

options.paved.docs.path: the path to the Sphinx folder (where the Makefile resides).

paved.docs.**rsync_docs**()
Upload the docs to a remote location via rsync.

options.paved.docs.rsync_location: the target location to rsync files to.

options.paved.docs.path: the path to the Sphinx folder (where the Makefile resides).

***options.paved.docs.build_rel*: the path of the documentation** build folder, relative to *options.paved.docs.path*.

paved.docs.**ghpages**()
Push Sphinx docs to github gh-pages branch.

1.Create file .nojekyll

2.Push the branch to origin/gh-pages after committing using ghp-import

Requirements:

- easy_install ghp-import

Options:

- *options.paved.docs.** is not used
- *options.sphinx.docroot* is used (default=docs)
- *options.sphinx.builddir* is used (default=.build)

> **Warning:** This will DESTROY your gh-pages branch. If you love it, you'll want to take backups before playing with this. This script assumes that gh-pages is 100% derivative. You should never edit files in your gh-pages branch by hand if you're using this script because you will lose your work.

paved.docs.**showhtml**()
Open your web browser and display the generated html documentation.

3.5 paved.django

paved.django – common tasks for django projects.

paved.django.**manage**(*args*)
Run the provided commands against Django's manage.py

***options.paved.django.settings*: the dotted path to the django** project module containing settings.

***options.paved.django.manage_py*: the path where the django** project's *manage.py* resides.

paved.django.**call_manage**(*cmd*, *capture=False*, *ignore_error=False*)
Utility function to run commands against Django's *django-admin.py*/*manage.py*.

***options.paved.django.project*: the path to the django project** files (where *settings.py* typically resides). Will fall back to a DJANGO_SETTINGS_MODULE environment variable.

***options.paved.django.manage_py*: the path where the django** project's *manage.py* resides.

paved.django.**djtest**(*args*)
Run tests. Shorthand for *paver manage test*.

paved.django.**syncdb**(*args*)
Update the database with model schema. Shorthand for *paver manage syncdb*.

paved.django.**shell**(*info*)
Run the ipython shell. Shorthand for *paver manage shell*.

Uses *django_extensions* <*http://pypi.python.org/pypi/django-extensions/0.5*>, if available, to provide *shell_plus*.

paved.django.**start**(*info*)
Run the dev server.

Uses *django_extensions* <*http://pypi.python.org/pypi/django-extensions/0.5*>, if available, to provide *runserver_plus*.

Set the command to use with *options.paved.django.runserver* Set the port to use with *options.paved.django.runserver_port*

3.6 paved.util

paved.util – helper functions.

paved.util.**pip_install**(**args*)
Send the given arguments to *pip install*.

paved.util.**easy_install**(**args*)
Send the given arguments to *easy_install*.

paved.util.**rmFilePatterns**(**patterns*, ***kwargs*)
Remove all files under the given path with the given patterns.

paved.util.**rmDirPatterns**(**patterns*, ***kwargs*)
Remove all directories under the current path with the given patterns.

paved.util.**shv**(*command*, *capture=False*, *ignore_error=False*, *cwd=None*)
Run the given command inside the virtual environment, if available:

paved.util.**update**(*dst*, *src*)
Recursively update the destination dict-like object with the source dict-like object.

Useful for merging options and Bunches together!

Based on: http://code.activestate.com/recipes/499335-recursively-update-a-dictionary-without-hitting-py/#c1

3.7 paved.pycheck

pycheck – check python code.

paved.pycheck.**pycheckall**()
All pycheck tasks.

paved.pycheck.**sloccount**()
Print "Source Lines of Code" and export to file.

Export is hudson plugin compatible: sloccount.sc

requirements:

- sloccount should be installed.
- tee and pipes are used

options.paved.pycheck.sloccount.param

paved.pycheck.**findimports**()
print python module dependencies by findimports.

requirements:

- findimports should be installed. `easy_install findimports`

options.paved.pycheck.findimports.param

paved.pycheck.**pyflakes**()
passive check of python programs by pyflakes.

requirements:

- pyflakes should be installed. `easy_install pyflakes`

options.paved.pycheck.pyflakes.param

paved.pycheck.**pychecker**()
check of python programs by pychecker.

requirements:

- pychecker should be installed.

options.paved.pycheck.pychecker.param

paved.pycheck.**nose**()
Run unit tests using nosetests.

requirements:

- nose should be installed.

options.paved.pycheck.nose.param

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

p