envdir Documentation

Release 0.7

Jannis Leidel and contributors

Mar 26, 2018

Contents

This is a Python port of daemontools' tool envdir. It works on Windows and other systems which can run Python. It's well tested and doesn't need a compiler to be installed.

envdir runs another program with a modified environment according to files in a specified directory.

So for example, imagine a software you want to run on a server but don't want to leave certain configuration variables embedded in the program's source code. A common pattern to solve this problem is to use environment variables to separate configuration from code.

envdir allows you to set a series of environment variables at once to simplify maintaining complicated environments, for example in which you have multiple sets of those configuration variables depending on the infrastructure you run your program on (e.g. Windows vs. Linux, Staging vs. Production, Old system vs. New system etc).

Let's have a look at a typical envdir:

```
$ tree envs/prod/
envs/prod/
├── DJANGO_SETTINGS_MODULE
├── MYSITE_DEBUG
├── MYSITE_DEPLOY_DIR
├── MYSITE_SECRET_KEY
└── PYTHONSTARTUP

0 directories, 3 files
$ cat envs/prod/DJANGO_SETTINGS_MODULE
mysite.settings
$
```

As you can see each file has a capitalized name and contains the value of the environment variable to set when running your program. To use it, simply prefix the call to your program with envdir:

```
$ envdir envs/prod/ python manage.py runserver
```

That's it, nothing more and nothing less. The way you structure your envdir is left to you but can easily match your configuration requirements and integrate with other configuration systems. envdirs contain just files after all.

An interesting summary about why it's good to store configuration values in environment variables can be found on the 12factor site.

Note: This Python port behaves different for multi line environment variables. It will not only read the first line of the file but the whole file. Take care with big files!

Tip: Feel free to open tickets at https://github.com/jezdez/envdir/issues.

CHAPTER 1

More documentation

1.1 Installation

1.1.1 As Python package

```
$ pip install envdir
```

or:

```
$ easy_install envdir
```

1.1.2 As standalone script

Alternatively you can also download a standalone executable that follows Python's PEP 441 and works with the Python Launcher for Windows (PEP 397). Simply install the launcher from its site (downloads) and you're ready to follow the rest of the instructions below.

Windows

Note: The Python Launcher for Windows also provides other useful features like being able to correctly launch Python when double clicking a file with the .py file extension, a `py` command line tool to easily launch the interactive Python shell when you're working on the command line. See the Python Launcher for Windows documentation for more infos.

Next step is downloading the actual standalone script. On Windows this entails using your web browser to download the following URL:

```
https://github.com/jezdez/envdir/releases/download/0.7/envdir-0.7.pyz
```

Or simply run this on the command line to trigger the download with your default web browser:

```
C:\WindowsExplorer.exe https://github.com/jezdez/envdir/releases/download/0.7/
↪envdir-0.7.pyz
```

Then – from the location you downloaded the file to – run the envdir script like you would any other script:

```
C:\Users\jezdez\Desktop>.\envdir-0.7.pyz ..
```

Linux, Mac OS, others

On Linux, Mac OS and other platforms with a shell like bash simply download the standalone file from Github:

```
$ curl -LO https://github.com/jezdez/envdir/releases/download/0.7/envdir-0.7.
↪pyz
```

and then run the file like you would do when running the script installed by the envdir package (see above):

```
$ ./envdir-0.7.pyz ..
```

1.2 Usage

1.2.1 Command line

From the original envdir documentation:

> envdir runs another program with environment modified according to files in a specified directory.
>
> Interface:
>
> ```
> envdir d child
> ```
>
> `d` is a single argument. `child` consists of one or more arguments.
>
> envdir sets various environment variables as specified by files in the directory named `d`. It then runs `child`.
>
> If `d` contains a file named `s` whose first line is `t`, envdir removes an environment variable named `s` if one exists, and then adds an environment variable named `s` with value `t`. The name `s` must not contain `=`. Spaces and tabs at the end of `t` are removed. Nulls in `t` are changed to newlines in the environment variable.
>
> If the file `s` is completely empty (0 bytes long), envdir removes an environment variable named `s` if one exists, without adding a new variable.
>
> envdir exits `111` if it has trouble reading `d`, if it runs out of memory for environment variables, or if it cannot run child. Otherwise its exit code is the same as that of child.

Alternatively you can also use the `python -m envdir` form to call envdir.

1.2.2 Isolated shell

envdir also includes an optional CLI tool called `envshell` which launches a subshell using the given directory. It basically allows you to make a set of environment variable stick to your current shell session if you happen to use envdir a lot outside of simple script use.

For example:

```
$ envshell ~/mysite/envs/prod/
Launching envshell for /home/jezdez/mysite/envs/prod. Type 'exit' or 'Ctrl+D' to␣
↪return.
$ python manage.py runserver
..
```

To leave the subshell, simply use the `exit` command or press `Ctrl+D`.

1.2.3 Setup an empty environment variable

Use an empty line to setup an empty environment variable (in contrast to an empty file, which would unset the environment variable):

```
$ echo > envdir/EMPTY_ENV
$ envdir envdir env | grep EMPTY_ENV
EMPTY_ENV=
```

1.3 Python API

`envdir.`**`open`**`(`[*path*]`)`

To use an envdir in a Python file (e.g. Django's `manage.py`) simply call the `open` function of the envdir module:

```
import envdir
envdir.open()
```

envdir will try to find an `envdir` directory next to the file you modified.

It's also possible to explicitly pass the path to the envdir:

```
import envdir

envdir.open('/home/jezdez/mysite/envs/prod')
```

Calling `open` will automatically apply all environment variables to the current instance of `os.environ`.

If you want to implement more advanced access to envdirs there is also an own dict-like *`Env`* object to work with. The above example could also be written like this:

```
import envdir

env = envdir.open('/home/jezdez/mysite/envs/prod')
```

The returned *`Env`* instance has a dict-like interface but also features a *`clear()`* method to reset the current instance of `os.environ` to the value it had before the envdir was opened:

```
import envdir

env = envdir.open('/home/jezdez/mysite/envs/prod')

# do something

env.clear()
```

Since calling the clear method should be done in a transparent manner you can also use it as a context manager:

```
import envdir

with envdir.open('/home/jezdez/mysite/envs/prod') as env:
    # do something
```

Outside the context manager block the environ is reset back automatically.

To access and write values you can also use the dict-like interface:

```
import envdir

with envdir.open() as env:
    env['DATABASE_URL'] = 'sqlite://:memory:'
    assert 'DATABASE_URL' in env
    assert env.items() == [('DATABASE_URL', 'sqlite://:memory:')]
```

Note: Additions to the envdir done inside the context manager block are persisted to disk and will be available the next time your open the envdir again.

Of course you can also directly interact with `Env` instances, e.g.:

```
import envdir

with envdir.Env('/home/jezdez/mysite/envs/prod') as env:
    # do something here
```

The difference between instantiating an `Env` yourself to using `envdir.open()` is that you'll lose the automatic discovery of the `envdir` directory.

See the API docs below for a full list of methods available in the `Env` object.

class `envdir.`**`Env`**(*path*)

An dict-like object to represent an envdir environment with extensive API, can be used as context manager, too.

`__cmp__`(*dict*)

`__contains__`(*name*)

`__delitem__`(*name*)

`__enter__`()

`__exit__`(*type*, *value*, *traceback*)

`__getitem__`(*name*, *default=<object object>*)

`__hash__ = None`

`__init__`(*path*)

`__iter__`()

`__len__`()

`__module__ = 'envdir.env'`

`__repr__`()

`__setitem__`(*name*, *value*)

clear()
Clears the envdir by resetting the os.environ items to the values it had before opening this envdir (or removing them if they didn't exist). Doesn't delete the envdir files.

copy()

fromkeys(*iterable*, *value=None*)

get(*key*, *failobj=None*)

has_key(*key*)

items()

iteritems()

iterkeys()

itervalues()

keys()

pop(*key*, **args*)

popitem()

setdefault(*key*, *failobj=None*)

update(**args*, ***kwargs*)

values()

1.4 Changelog

1.4.1 1.0.0 (26/03/2018)

- Drop python 2.6, 3.2 and 3.3
- Add explicit support for python 3.6
- Add support for symlinks
- Improved support for windows

1.4.2 0.7 (08/10/2014)

- Use *exec* (*os.execvpe*) to replace the envdir process with the child process (fixes #20).
- Change *isenvvar()* to only check for = in var names.

1.4.3 0.6.1 (12/23/2013)

- Fixed handling SIGTERM signals to make sure all children of the forked process are killed, too. Thanks to Horst Gutmann for the report and help fixing it.

1.4.4 0.6 (12/03/2013)

- Rewrote tests with pytest.
- Vastly extended Python API.
- Added Sphinx based docs: https://envdir.readthedocs.io/
- Fixed killing child process when capturing keyboard interrupt.
- Added standalone script based on PEPs 441 and 397, compatible with Python Launcher for Windows. See the installation instructions for more info.

1.4.5 0.5 (09/22/2013)

- Added check if the the provided path is a directory and throw an error if not. This adds compatibility to the daemontools' envdir.
- Make sure to convert Nulls (`\0`) to newlines as done so in daemontools' envdir.

1.4.6 0.4.1 (08/21/2013)

- Fixed `envdir.read()` to actually work with already existing environment variables. Extended docs to test Python use.

1.4.7 0.4 (08/09/2013)

- Added `envshell` command which launches a subshell using the environment as defined in the given envdir. Example:

```
$ envshell ~/mysite/envs/prod/
Launching envshell for /home/jezdez/mysite/envs/prod. Type 'exit' or 'Ctrl+D' to␣
↪return.
$ python manage.py runserver
..
```

1.4.8 0.3 (07/30/2013)

- Catch `KeyboardInterrupt` exceptions to not show a traceback from envdir but the repsonse from the called command.
- Allow multiline environment variables. Thanks to Horst Gutmann for the suggestion. This is a departure from daemontools' standard which only allows the first line of the environment variable file.

1.4.9 0.2.1 (07/11/2013)

- Fixed `python -m envdir`
- Extended README to better describe the purpose

1.4.10 0.2 (07/10/2013)

- Added ability to use envdir from Python.

1.4.11 0.1 (07/10/2013)

- Initial release.

Index