Plant Documentation

Release 0.1.3

Gabriel Falcão

Sep 08, 2017

Contents

Contents:

CHAPTER 1

Introduction

Plant is a tiny python library that provide handy functions for path manipulation, file search, and other filesystem-based I/O operations.

It's called plant because you start using an instance of `Node` and with search operations you start moving through other nodes that represent paths, they can be folders or files, in fact every possible path in the disk your software is operating on is a potential "plant" node. *(Though it's also given my personal affection towards plants and vegetables)*

1.1 Primer

A `Node` takes a path, if it's relative, Plant will turn it into absolute before storing it internally.

Plant always has the absolute path of the current node. (although below you can see that for debugging purposes the string representation of a node shows the relative path since the absolute might be really long.

```
>>> from plant import Node
>>>
>>> unit_test_folder = Node("tests/unit")
>>> unit_test_folder
Node(path='tests/unit')
```

CHAPTER 2

Every node is a file

And because in unix a directory is a folder, a node might also be a directory.

```
>>> from plant import Node
>>>
>>> functional_test_file = Node("tests/functional/test_fs.py")
>>> functional_test_file.is_dir
False
>>> functional_test_file.is_file
True
>>> functional_test_file.parent
Node('tests/functional')
>>> functional_test_file.parent.parent
Node('tests')
>>> functional_test_file.dir
Node('tests/functional')
```

2.1 Handy way to get to a directory from a file

`.dir` is a safe way to be in the current working directory

```
>>> from plant import Node
>>>
>>> unit_test_file = Node("tests/unit/test_base.py")
>>> unit_test_file.dir
Node('tests/unit')
>>> unit_test_file.dir.dir
Node('tests/unit')
>>> unit_test_file.dir.dir.dir
Node('tests/unit')
```

CHAPTER 3

Finding files that match a certain regex

Notice that it is recursive

```
>>> from plant import Node
>>>
>>> test_files = Node("tests").find_with_regex("test_.*.py")
>>> test_files
[Node('tests/functional/test_fs.py'), Node('tests/unit/test_base.py'), Node('tests/
↪unit/test_node.py')]
```

CHAPTER 4

Finding only the first occurrence

Very handy for finding one file at a time

```
>>> from plant import Node
>>>
>>> found = Node("tests").find("test_base.py")
>>> found
Node('tests/unit/test_base.py')
```

CHAPTER 5

String operations on a Node

A node has many handy properties and methods, you can get the relative path to a certain file, get the base name, and some other sweets:

```
>>> from plant import Node
>>>
>>> unit_test_file = Node("tests/unit/test_base.py")
>>> unit_test_file.basename
u'test_base.py'
>>> unit_test_file.path
u'/.../tests/unit/test_base.py'
```

5.1 Aso works with directories

```
>>> from plant import Node
>>>
>>> unit_test_file = Node("tests/unit/test_base.py")
>>> unit_test_file.dir.basename
u'unit'
```

CHAPTER 6

API Reference

class plant.**Node**(*path*)

Node is a file abstraction.

The constructor takes a path as a parameter and grabs filesystem information about it.

Its attributes *is_file* and *isdir* are booleans and are useful for quickly identifying its 'type', which among Plant's engine codebase is either 'blob', for a file and 'dir' for a directory.

It also has *self.metadata*, which is just a handy *DotDict* containing the results of calling *os.stat* (mode, ino, dev, nlink, uid, giu, size, atime, mtime, ctime)

basename

extracts the basename the node path

```
>>> from plant import Node
>>>
>>> Node('/srv/application/conf.py').basename
'conf.py'
```

Returns bytes

cd(*path*)

Shortcut for `Node.goto()`, also works for files, though has a better semantic outlook when handling directories.

```
>>> from plant import Node
>>>
>>> Node('/opt/media/mp3').cd('../mp4')
Node('/opt/media/mp4')
```

Parameters **path** – bytes

Returns a Node

contains(*path*)
Checks if the given path exists as an immediate relative of the current node's path

```
>>> from plant import Node
>>>
>>> Node('/opt/media/mp3').contains('unknown-track.mp3')
False
>>>
>>> Node('/opt/media/mp4').contains('../mp3/music1.mp3')
True
```

Parameters **path** – bytes

Returns bool

could_be_updated_by(*other*)
check to see if the mtime of another Node is greater than the current one.

Parameters **other** – the other Node

Returns bool - true if the other node is "newer"

depth_of(*path*)
Calculates the level of depth of the given path inside of the instance's path.

Only really works with paths that are relative to the class.

```
>>> from plant import Node
>>>
>>> level = Node('/foo/bar').depth_of('/foo/bar/another/dir/file.py')
>>> level
2
```

Parameters **path** – bytes

Returns a bool

dir
returns a Node pointing to the current directory.

this call is idempotent in the sense that chaining up multiple class results in the same node.

```
>>> from plant import Node
>>>
>>> Node('/srv/application/conf.py').dir
Node('/srv/application')
>>>
>>> Node('/srv/application/conf.py').dir.dir
Node('/srv/application')
>>>
>>> Node('/srv/application/conf.py').dir.dir.dir
Node('/srv/application')
>>>
>>> # and so on
```

Returns Node

directory
shortcut for `Node.dir`

```
>>> from plant import Node
>>>
>>> Node('/srv/application/conf.py').directory
Node('/srv/application')
>>>
>>> Node('/srv/application/conf.py').directory.directory
Node('/srv/application')
>>>
>>> Node('/srv/application/conf.py').directory.directory.directory
Node('/srv/application')
>>>
>>> # and so on
```

Returns `Node`

find(*relative_path*)
Calls `Node.find_with_regex()` with `lazy=True` but only returns the first occurrence.

```
>>> from plant import Node
>>>
>>> Node('/opt/media').find('[.](mp3|mp4)$')
Node('/opt/media/mp3/music1.mp3')
```

Parameters **relative_path** – `bytes`

Returns a `Node`

find_with_regex(*pattern*, *flags=0*, *lazy=False*)
searches recursively for children that match the given regex returning a respective [python‘Node‘] instance for that given.

It works like `Node.glob()` but applies a regexp match rather instead.

```
>>> from plant import Node
>>>
>>> Node('/opt/media').find_with_regex('[.](mp3|mp4)$')
[
    Node('/opt/media/mp3/music1.mp3'),
    Node('/opt/media/mp3/music2.mp3'),
    Node('/opt/media/mp4/my-video.mp4'),
 ]
```

Parameters

- **pattern** – a valid `fnmatch` pattern string
- **lazy** – bool - if True returns an iterator, defaults to a flat `list`

Returns an iterator or a list of `Node`

glob(*pattern*, *lazy=False*)
searches for globs recursively in all the children node of the current node returning a respective [python‘Node‘] instance for that given.

Under the hood it applies the given `pattern` into `fnmatch.fnmatch()`

```
>>> from plant import Node
>>>
>>> mp3_node = Node('/opt/media/mp3')
>>> mp3_node.glob('*.mp3')
[
    Node('/opt/media/mp3/music1.mp3'),
    Node('/opt/media/mp3/music2.mp3'),
 ]
```

Parameters

- **pattern** – a valid `fnmatch` pattern string
- **lazy** – bool - if True returns an iterator, defaults to a flat `list`

Returns an iterator or a list of `Node`

goto(*path*)

Returns a `Node` pointing to the given directory

```
>>> from plant import Node
>>>
>>> Node('/opt/media/mp3/').goto('../mp4/my-video.mp4')
Node('/opt/media/mp4/my-video.mp4')
```

Parameters **path** – `bytes`

Returns a `Node`

join(**path*)

Joins the given path with that of the current node's

Does not check if the target file exists, it simply concatenates strings using the native platform's path separator.

```
>>> from plant import Node
>>>
>>> Node('/opt/media/mp3').join('unknown-track.mp3')
'/opt/media/mp3/unknown-track.mp3'

>>> Node('/opt/media/mp4').join('..' , 'mp3', 'music1.mp3')
'/opt/media/mp3/music1.mp3'
```

Parameters **path** – `bytes`

Returns `bytes`

list()

returns a list of files children of the current directory if the node points to a file, it's siblings will be listed

```
>>> from plant import Node
>>>
>>> Node('/srv/application/conf.py').list()
[
    Node('/srv/application/conf.py'),
    Node('/srv/application/README.rst')
]
```

Returns a `list` of `Node`

classmethod new(**args*, ***kw*)

creates a new instance of `Node` mostly used internally by its methods.

Parameters

- ***args** –
- ****kw** –

Returns a new instance of `Node`

open(*path*, **args*, ***kw*)

performs an `io.open()` on the given relative path to the current node.

```
>>> from plant import Node
>>>
>>> documents = Node('/opt/documents')

>>> with documents.open('hello-world.txt', 'wb', 'utf-8') as f:
...     f.write('HELLO WORLD')

>>> documents.open('hello-world.txt').read()
'HELLO WORLD'
```

Parameters

- **path** – `bytes`
- ***args** – passed onto `io.open()`
- ***kw** – passed onto `io.open()`

Returns `io.FileIO`

parent

returns a `Node` pointing to the parent directory.

it stops at the root node.

```
>>> from plant import Node
>>>
>>> Node('/srv/application').parent
Node('/srv')
>>>
>>> Node('/srv/application').parent.parent
Node('/')
>>>
>>> # it stops at the root node
>>> Node('/srv/application').parent.parent.parent
Node('/')
```

Returns `Node`

path_to_related(*path*)

Returns the path to a related file. (is under a subtree the same tree as the node).

It's useful to know how to go back to the root of this node instance.

```
>>> from plant import Node
>>>
>>> way_back = Node('/foo/bar').path_to_related('/foo/bar/another/dir/file.py
↪')
>>> way_back
'../../'

>>> way_back = Node('/foo/bar/docs/static/file.css').path_to_related('/foo/
↪bar/docs/intro/index.md')
>>> way_back
'../static/file.css'
```

Parameters **path** – bytes

Returns a bytes

relative(*path*)

returns the relative from the current Node to the given string

```
>>> from plant import Node
>>>
>>> Node('/opt/media/mp3/').relative('/opt/media/mp4/my-video.mp4')
'../mp4/my-video.mp4'
```

Parameters **path** – a path string

Returns bytes

trip_at(*path*, *lazy=False*)

Iterates recursively on a subpath of the current Node

It basically performs a py:func:*os.walk* at the given path and yields the absolute path to each file

```
>>> from plant import Node
>>>
>>> Node('/opt/media').trip_at('mp3', lazy=False)
[
    '/opt/media/mp3/music1.mp3',
    '/opt/media/mp3/music2.mp3',
 ]
```

Parameters

- **path** – a path string
- **lazy** – bool - if True returns an iterator, defaults to a flat list

Returns an iterator or a list of bytes

walk(*lazy=False*)

Same as Node.trip_at() but iterates recursively within the current Node instead.

```
>>> from plant import Node
>>>
>>> Node('/opt/media').walk(lazy=False)
[
    '/opt/media/mp3/music1.mp3',
    '/opt/media/mp3/music2.mp3',
```

```
    '/opt/media/mp4/my-video.mp4',
]
```

Parameters

- **path** – a path string
- **lazy** – bool - if True returns an iterator, defaults to a flat `list`

Returns an iterator or a list of `bytes`

`plant.handy.`**slugify**(*text*, *repchar=u'-'*)

takes a string and replaces all non-alphanumeric characters with the given `repchar`.

Parameters

- **text** – the string to be slugified
- **repchar** – defaults to –, the replacement char

Returns **:py:'bytes'**

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index