FsQuass Documentation

Release 0.1.0

Dmitri Lebedev

October 22, 2012

CONTENTS

CHAPTER

ONE

WHY ANOTHER FILESYSTEM LIBRARY?

None of them lets you work with files and paths in an elegant way. By elegant I mean as jQuery lets you work with DOM. A jQuery for filesystem is not a new idea, though: someone has written *fsquery* library for Node.js. There is a good pyFileSystem library that supports different file systems (like ssh, memory, ftp).

I rather needed a path traversing tool than filesystem.

Let's say, I want to find photos (`*.jp[e]g`) in my pictures folders for 2010 and 2011. The folders for those days end with `'repairs'`.

```
from fsquass import Fs
photos = Fs('/home/siberiano/Pictures/{2011,2010} *repairs *.jp?g:ignorecase')
```

Note how syntax is similar to what we're used to in file systems:

- spaces in path stand for descendant, like in CSS
- curly braces `{x,y}` mean different masks of a name
- stars and question marks work like in command line
- pseudo-classes (`:ignorecase`) work like in CSS

Now I can proceed, say, find the folders of these files:

```
photos.parents().find('*.txt')
photos.filter('DCIM*')
```

As you can see, it's quite similar to jQuery. But unlike jQuery, Python's set classes support set operations:

```
photos.parents().children() - photos
```

Everything you can do with `set`, you can do with `Fs` objects as well.

CHAPTER

TWO

INSTALLATION

Clone it and install with setuptools:

```
hg clone https://bitbucket.org/siberiano/fsquass
cd fsquass
python setup.py install
```

CHAPTER

THREE

WHAT DOES THE NAME MEAN?

It's called so to give it a unique name and flavour. 'Quass' has common part with 'query', but means kvass.

CHAPTER

FOUR

CONTENTS

4.1 Quick Tour

4.1.1 As Commandline Tool

Once you install FsQuass, it can work as command line tool. It's almost like GNU find, except that it *only searches* and does not try to be a Swiss knife. There are no options in it:

```
$ fsquass '/home/*'
/home/siberiano
/home/guest
```

note It's better to quote the arguments since Bash may try to convert masks (`*`, `.`) for you.

If you need to do something with the found files, use `xargs`:

```
$ fsquass '/home .bashrc' | xargs cat
```

This prints the contents of all .bashrc files *Descendants* of user folders. As a quick tip on xargs, to pass file path in the middle of a command, use curly braces:

```
$ fsquass '/home .bashrc' | xargs -I '{}' ln -S {} /tmp
```

To each found `.bashrc` this will make a symlink in `/tmp`.

4.1.2 As a Python Module

The `Fs` class (stands for *files set*), like jQuery, searches by string and also inherits the API of `set` class with all set operations: union, intersection, add, remove, etc.

```
from fsquass import Fs
Fs('/home .bashrc') - Fs('~/.bashrc')  # similar to jQuery.not()
```

File sets are iterable and consist of `File` or `Dir` instances. They also can generate strings:

```
for project in Fs('~/projects/*'):
        print project

for path in Fs('~/projects/*'):  # a generator of string paths
        print path
```

4.1.3 Syntax

The syntax is essentially Unix filename patterns + some powerful extensions. Patterns work via `fnmatch` module. The special characters used in shell-style wildcards are: `*`, `?` (any single character), `[abc]` (a, b or c), `[!abc]` anything but them. Some simple examples:

```
folder/folder/file.py[co]
folder/*/*.txt
/etc/hosts
/var/log/*.log
./file
file
../another_folder/file
```

The two latter examples are the same.

But here come some extensions: you can go a level up from a file:

```
fsquass/setup.py/..
```

This expression will evaluate to `fsquass`, but only if `setup.py` is present. This is useful if you need folders to contain specific children.

Descendants

It works like in CSS:

```
~ *.py
```

will search for `*.py` anywhere in the home folder, at any folders depth.

> **attention** This kind of search is expensive since it makes the program go through all the directory tree down from `~`. Make such searches as narrow as possible if you can: `*/projects/django *.py`.

You can make several such searches:

```
~/projects templates *.haml
```

Scans `projects` folder for `templates`, then scans each of those for `*.haml`.

The second part of descendant can be multi-level:

```
~/projects templates/*.haml
```

Use backslash to write a space in a name. Use double backslash if you need to escape the backslash itself:

```
~/project\ description/*
```

Yet there is no syntax for the opposite search, for random number of levels upwards.

Pseudo-Classes

Similar to those in CSS, they are written in the end or instead of a pattern, and either filter filesystem objects by type, or modify the pattern's properties:

Pseudo-Class	Meaning
`:file`	object is a file
`:dir`	object is a directory
`:ignorecase`	makes search case-insensitive

Examples:

```
~/.*:dir
~/*:file
/home/:dir
/:dir
~/Pictures *.jpg:ignorecase
```

Sub-Patterns

In Unix shell, you can do this: `mkdir project/{apps,templates,static}`. The same works in fsquass. Between slashes, you can use curly braces to write multiple options:

```
/home/{siberiano,guest}/.bashrc
~/Pictures {*.jpeg:ignorecase,*.jpg:ignorecase}
```

> **note** Pseudo-classes must be inside the curly braces.

Multiple Patterns

If you need to find objects with completely different paths or patterns, write multiple expressions separated with a colon:

```
/etc/hosts;~/.my_hosts;~/test.txt
```

Put a backslash to a colon if it's a part of a name:

```
strange\;name1;strange\;name2
```

4.1.4 Set Operations

`Fs` inherits from `set` and suspports all the set methods.

```
Fs('./*.py') | Fs('./*.pyc')  # union
Fs('. {*.rst,*.txt}') - Fs('./build *.txt')  # not
Fs('*.py') & Fs('__*__.py')  # intersection
Fs('*.py') ^ Fs('__*__.py')  # xor (union not intersection)
```

`filter` is just like interection, but is faster since it doesn't search files on disk.

```
Fs('*.py').filter('__*__.py')
```

4.1.5 Traversing

Having one set of files you can generate another set relative of it:

```
# find Python scripts and then their parent folders that start with 'django'
django_projects = Fs('~/projcets *.py').parents('django*')

# inside those find __init__.py at top level
django_projects.children('__init__.py')

# or at any depth
django_projects.find('. __init__.py')
```

```
# find doc roots inside them, by relative path (no recursive search)
django_projects.find('docs/source/index.*')

django_projects.siblings()
```

4.1.6 Files Manipulation

Currently `Fs` supports

- `linkTo`
- `symlinkTo`

4.2 `fsquass` Package

FsQuass is a filesystem query and traversing library, a pythonic jQuery for filesystem.

Still work in progress.

class `fsquass.__init__.Fs`(*nodes=None*)

Bases: `set`

Files set. Is a `set` of `File` and `Dir` instances with traversal methods. Besides the methods inherited from `set`, it has some methods and properties specific to file systems.

nodes can be a string or an iterable. of `File` and `Dir` instances.

If *nodes* is a string, it's treated differently depending on what it starts with:

- `/`, files are matched from those in the root directory and further, without scanning the whole filesystem.
- `./`, the next name will be searched inside the current directory, without recursive scanning.
- `~`, home folder will be opened
- `~/`, home folder will be opened, and it's children will be matched, without recursive scanning.

A space is treated like in CSS, a recursive search for descendants. E.g.

```
Fs('/home/user tests/__init__.py')
```

will

- find `/home/user`,
- then recursively scan both for files and directories named `tests`,
- then will search for `__init__.py` inside those directories, but not deeper.

Note: Recursive scans can be expensive. If you

`children`(*pattern=None*)

Returns a set of children of all the set items filtered by *pattern*.

`closest`(*pattern*)

Finds the closest ancestors by pattern.

`exclude`(*pattern*)

Exclude items that match pattern.

filter(*pattern*)
Filters items_list by *patten*.

Filtering a set of paths is equal to an intersection of the set and of a set found by *pattern*:

```
dirs = Fs('/home/siberiano;/home;/tmp;/tmp/siberiano')
dirs.filter('siberiano') == dirs & Fs('/').find('. siberiano')
```

find(*pattern*)
Searches by *pattern* inside the set items. Returns a new Fs instance. E.g. if we have a set `fs` of these paths:

```
/home/user/
/root
```

`fs.find('.bashrc')` will probably output:

```
/home/user/.bashrc
/root/.bashrc
```

If you need to find multiple paths, separate them with semicolon:

```
Fs('/home/siberiano').find('.bashrc;Work/project/templates base.haml')
```

Will search for `.bashrc` file in my homefolder (but not deeper) and inside ~/Work/project/templates will recursively search for `base.haml` files.

To avoid accidental scanning of the entire filesystem, recursive search is made harder. Use dot and space in the beginning if you need it anyway:

```
# scan the entire filesystem for 'siberiano'
Fs('/').find('. siberiano')
# scans for files & directories named 'project' inside Work
Fs('/home/siberiano/Work').find('. project')
```

first()
Returns the first item from the set. A shortcut for `iter(fs).next()`

linkTo(*target*, *multiple_targets=False*, *name_callback=None*)
Makes a hard link to all the set members in *target* folder.

- target must be a set of 1 or more directories (`Dir` instances).
- if *multiple_targets* parameter is `True`, links will be made in all the *target* folders. If *multiple_targets* is `False`, then will link in the first *target* folder only.

Optional *name_callback* should work like this:

```
def name_callback(source, target):
        # source & target are Fs instances with 1 member each
        return source, target
```

parents()
Returns a set of parents of all the items, e.g. for

```
/home/user/.bashrc
/home/user/.hgrc
/tmp/test
/tmp
```

parents will be

```
/home/user
/tmp
/
```

paths
A generator of paths of all the items.

siblings (*pattern=None*)
Finds all the siblings of the files in set, filtered by *pattern*. The result will not include any files of the original set.

symlinkTo (*target*, *multiple_targets=False*, *name_callback=None*)
Makes a symbolic link in *target* folder like `Fs.linkTo()`

class `fsquass.__init__.`**Dir** (*full_path*)
Bases: `fsquass.__init__.File`

Directory. Returns its directories and files in children() method.

•Is iterable:

```
for i in Dir('/home/siberiano'):
        print i
```

will print files and directories in the folder.

This allows using such tricks as using a `Dir` to get a `Fs` of it's children:

```
>>> d = Dir('/')
>>> Fs(d) == d.children()
True
```

•Can check if contains another `File` or `Dir`:

```
>>> Dir('/home') in Dir('/')
True
>>> Dir('/tmp') in Dir('/home')
False
```

children (*pattern=None*)
Lists the directory and returns `Fs` of the files, filtered by *pattern*.

delete (*sure=False*)
Deletes the directory with all files and directories in it if *sure* is `True`. If you managed to call it like this, don't blame the library for any lost data.

open (**args*, ***kwargs*)
Raises TypeError, since directories can't be opened like files.

class `fsquass.__init__.`**File** (*full_path*)
Bases: `object`

A file or a directory. Contains self.path, and if an object with the same absolute path is instantiated, an existing item is returned. If *full_path* is unaccessible, `EnvironmentError` is raised.

basename
String basename of the file.

children (*pattern=None*)
Returns an Fs of child nodes. Makes sense in Dir only, but put here for compatibility.

delete(*sure=False*)
Deletes the file if *sure* is `True`. If you managed to call it like this, don't blame the library for any lost data.

open(**args*, ***kwargs*)
Wrapper to Python `open()`.

parent
Returns an Fs with the parent directory.

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

f