Formic Documentation

Release 1.0.3

Aviser LLP, Singapore

Jun 01, 2018

Contents

CHAPTER 1

Installing Formic

1.1 Prequisites

Platform and dependencies:

- Formic requires Python 2.6+ or Python3.4+
- It has been tested on
  - Mac OS X (Lion and Mountain Lion)
  - Ubuntu 11.10 and 12.04LTS
  - Windows XP, Windows 7 (Home Premium) and Windows 10.

Formic can work on any Python 2.6+ or Python 3.4+ system; if not, please contact the maintainer or file an issue.

Formic has no runtime dependencies outside the Python system libraries.

1.2 Installation options

There are three ways to obtain Formic shown below, in increasing difficulty and complexity. You need only pick one:

Option 1: Automated install

Simplest: use:

```
$ easy_install formic2
```

or:

```
$ pip install formic2
```

Option 2: Source install

1. Download the appropriate package from Formics page on the Python Package Index. This is a GZipped TAR file.
2. Extract the package using your preferred GZip utility.
3. Navigate into the extracted directory and perform the installation:

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

Option 3: Check out the project

If you like, you could download the source and compile it yourself. The source is now on GitHub.

After checking out the source, navigate to the top level directory and build:

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

1.3 Validating the installation

After installing, you should be able to execute Formic from the command line:

```
$ formic --version
```

If you downloaded the source, you can additionally run the unit tests. This requires py.test:

```
$ easy_install pytest
$ pytest -v test
```

1.4 Compiling the documentation

Formic uses Sphinx for documentation. The source files are in the 'doc' subdirectory. To build the documentation,

1. Ensure that formic has been installed and is visible on the path so you can start Python and import formic, eg:

```
$ cd /anywhere/on/filesystem
$ python
Python 2.7.1 (r271:86832, Jul 31 2011, 19:30:53)
[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2335.15.00)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import formic
>>> exit()
$
```

2. Navigate to Formic's top level directory, then:

```
$ sphinx-build doc htmlout
```

The documentation will be in the ./htmlout subdirectory.

Note: Only HTML generation has been tested.

Note: If you get errors that Sphinx cannot import Formic's packages, you may not have installed the module correctly. Try reinstalling it, eg 'python setup.py develop' or 'python setup.py install'

CHAPTER 2

Changelog

2.1 1.0.3

New feature:

- Fix #4, symlinks options now can work.
- Support ReadTheDocs .

2.2 1.0.2

New feature:

- Fix #2, FileSet now can contain arg casesensitive=True|False, default True on POSIX, but always False on NT.
- Fix #2, casesensitive can apply on both file-name and directory-name.

2.3 1.0.1

New feature:

- Add supports for both Python3 and Python2.
- Keep all the interface consistent with the 0.9beta8.

2.4 0.9beta8

New feature:

- Added the ability to pass in the function that walks the directory path, which allows for alternate implementations or supplying a mock function that provides values completely unrelated to the OS. This is available only from the API and not from the command line:

```
files = ["CVS/error.py", "silly/silly1.txt", "1/2/3.py", "silly/silly3.txt", "1/2/
↪4.py", "silly/silly3.txt"]
for dir, file in FileSet(include="*.py", walk=walk_from_list(files)):
    print dir, file
```

Bug fixes:

- Fixed #10: Paths like "//network/dir" caused an infinite loop
- Fixed #11: Incorrect handling of globs ending "/**" and "/". Ant Glob semantics for:

```
**/test/**
```

  are that they should match "all files that have a test element in their path, including test as a filename."

2.5 0.9beta7

Bug fixes:

- Fixed #4 and #6: Handles mixed case correctly on Windows
- Fixed #8: Formic fails if it starts directory traversal at any drive's root directory on Windows
- Fixed #5: Formic had an unnecessary dependency on pkg_resources

Improvements:

- Fixed performance defect #7: Much faster searching for globs like "/a/b/**"
- Improved quickstart documentation: Explicitly mention that Formic searches from the *current* directory.

2.6 0.9beta6

- Fixed issue #2: VERSION.txt was not being correctly packaged causing problems with source and pip installation
- Fixed issue #3: Incorrect behaviour when absolute directory was "/"
- Removed Google Analytics from documentation, and improved documentation template
- Improved publishing process.

2.7 0.9beta5

This is a documentation and SCM release. No API changes.

- Updated documentation, changelogs and installation instructions
- Removed Google Analytics from Sphinx documentation
- Implemented Dovetail build * Added coverage, pylint and sloccount metrics generation * Added command-line sanity tests

2.8 0.9beta4

- Fixed issue #1: In 3de0331450c0

2.9 0.9beta3

- API: FileSet is now a natural iterator:

```
fs = FileSet(include="*.py")
filenames = [ filename for filename in fs ]
```

- API: `__str__()` on Pattern and FileSet has been improved. Pattern now returns the just normalized string for the pattern (eg `**/*.py`). FileSet now returns the details of the set include all the include and exclude patterns.
- Setup: Refactored setup.py and configuration to use only setuptools (removing distribute and setuptools_hg)
- Documentation: Renamed all ReStructured Text files to .rst. Small improvements to installation instructions.

2.10 0.9beta2

- Refactored documentation files and locations to be more DRY:
  - Sphinx documentation
  - setup.py/Pypi readme
  - README/INSTALL/CHANGELOG/USAGE in topmost directory
- Removed the file-based distribute depending on explicit dependency in setup.py

2.11 0.9beta

Date: 14 Apr 2011 First public release

CHAPTER 3

Usage

3.1 Command Line

The formic command provides shell access to Ant glob functionality. Some examples are shown below.

Find all Python files under `myapp`:

```
$ formic myapp -i "*.py"
```

(Note that if a root directory is specified, it must come before the -i or -e)

Find all Python files under the current directory, but exclude `__init__.py` files:

```
$ formic -i "*.py" -e "__init__.py"
```

... and further refined by removing test directories and files:

```
$ formic -i "*.py" -e "__init__.py" "**/*test*/" "test_*"
```

This will search for files all Python files under the current directory excluding all *__init__.py* files, any file in directories whose name contains the word 'test', and any files that start *test_*.

Output from Formic is formatted like the Unix find command, and so can easily be combined with other executables, eg:

```
$ formic -i "**/*.bak" | xargs rm
```

... will delete all `.bak` files in or under the current directory (but excluding VCS directories such as `.svn` and `.hg`).

Full usage is documented in the formic.command package.

3.2 Library

The API provides the same functionality as the command-line but in a form more readily consumed by applications which need to gather collections of files efficiently.

The API is quite simple for normal use. The example below will gather all the Python files in and under the current working directory, but exclude all directories which contain 'test' in their name, and all files whose name starts with 'test_':

```
import formic
fileset = formic.FileSet(include="**.py",
                         exclude=["**/*test*/**", "test_*"]
                         )

for file_name in fileset:
    # Do something with file_name
    ...
```

A more detailed description can be found in the API.

CHAPTER 4

Ant Globs

Apache Ant fileset is documented at the Apache Ant project:

- http://ant.apache.org/manual/dirtasks.html#patterns

4.1 Examples

Ant Globs are like simple file globs (they use ? and * in the same way), but include powerful ways for selecting directories. The examples below use the Ant glob naming, so a leading slash represents the top of the search, *not* the root of the file system.

`*.py`

Selects every matching file anywhere in the whole tree Matches `/foo.py` and `/bar/foo.py` but not `/foo.pyc` or `/bar/foo.pyc/`

`/*.py` Selects every matching file in the root of the directory (but no deeper).

Matches `/foo.py` but not `/bar/foo.py`

`/myapp/**` Matches all files under `/myapp` and below.

`/myapp/**/__init__.py` Matches all `__init__.py` files `/myapp` and below.

`dir1/__init__.py` Selects every `__init__.py` in directory `dir1`. `dir1` directory can be anywhere in the directory tree

Matches `/dir1/file.py`, `/dir3/dir1/file.py` and `/dir3/dir2/dir1/file.py` but not `/dir1/another/__init__.py`.

`**/dir1/__init__.py` Same as above.

`/**/dir1/__init__.py` Same as above.

`/myapp/**/dir1/__init__.py` Selects every `__init__.py` in dir1 in the directory tree `/myapp` under the root.

Matches `/myapp/dir1/__init__.py` and `/myapp/dir2/dir1/__init__.py` but not `/myapp/file.txt` and `/dir1/file.txt`

4.2 Default excludes

Ant FileSet (and Formic) has built-in patterns to screen out a lot of development 'noise', such as hidden VCS files and directories. The full list is at *get_initial_default_excludes()*.

Default excludes can be simply switched off on both the command line and the API, for example:

```
$ formic -i "*.py" -e "__init__.py" "**/*test*/" "test_*" --no-default-excludes
```

CHAPTER 5

API

5.1 `formic` Package

An implementation of Apache Ant globs.

- The *formic.formic* module contains the main class *FileSet*
- The *formic.command* module contains the command-line interface.

class formic.__init__.**FileSet**(*include*, *exclude=None*, *directory=None*, *default_excludes=True*, *walk=None*, *symlinks=True*, *casesensitive=True*)

Bases: object

An implementation of the Ant FileSet class.

Arguments to the constructor:

1. *include*: An Ant glob or list of Ant globs for matching files to include in the response. Ant globs can be specified either:
   (a) As a string, eg "*.py", or
   (b) As a *Pattern* object
2. *exclude*: Specified in the same was as *include*, but any file that matches an exclude glob will be excluded from the result.
3. *directory*: The directory from which to start the search; if None, the current working directory is used
4. *default_excludes*: A boolean; if True (or omitted) the *DEFAULT_EXCLUDES* will be combined with the *exclude*. If False, the only excludes used are those in the excludes argument
5. *symlinks*: Sets whether symbolic links are included in the results or not. Defaults to True.
6. *walk*: A function whose argument is a single directory that returns a list of (dirname, subdirectoryNames, fileNames) tuples with the same semantics of os.walk(). Defaults to os.walk()
7. *casesensitive*: Only effective on POSIX, default True. Always False on NT.

Implementation notes:

- *FileSet* is lazy: The files in the *FileSet* are resolved at the time the iterator is looped over. This means that it is very fast to set up and (can be) computationally expensive only when results are obtained.
- You can iterate over the same *FileSet* instance as many times as you want. Because the results are computed as you iterate over the object, each separate iteration can return different results, eg if the file system has changed.
- *include* and *exclude* arguments to the constructor can be given in several ways:
  - A string: This will be automatically turned into a *Pattern*
  - A *Pattern*: If you prefer to construct the pattern yourself
  - A list of strings and/or *Pattern* instances (as above)
- In addition to Apache Ant's default excludes, *FileSet* excludes:
  - `__pycache__`
- You can modify the *DEFAULT_EXCLUDES* class member (it is a list of *Pattern* instances). Doing so will modify the behaviour of all instances of *FileSet* using default excludes.
- You can provide and alternate function to `os.walk()` that, for example, heavily truncates the files and directories being searched or returns files and directories that don't even exist on the file system. This can be useful for testing or even for passing the results of one FileSet result as the search path of a second. See `formic.treewalk.walk_from_list()`:

```
   files = ["CVS/error.py", "silly/silly1.txt", "1/2/3.py", "silly/silly3.txt
↪", "1/2/4.py", "silly/silly3.txt"]
   fileset = FileSet(include="*.py", walk=treewalk.walk_from_list(files))
   for dir, file in fileset:
       print dir, file

This lists 1/2/3.py and 1/2/4.py no matter what the contents of the
current directory are. CVS/error.py is not listed because of the default
excludes.
```

DEFAULT_EXCLUDES = [/__pycache__/**/*, **/*~, **/#*#, **/.#*, **/%*%, **/._*, **/CVS**
Default excludes shared by all instances. The member is a list of *Pattern* instances. You may modify this member at run time to modify the behaviour of all instances.

files()
A generator function for iterating over the individual files of the FileSet.

The generator yields a tuple of `(rel_dir_name, file_name)`:

1. *rel_dir_name*: The path relative to the starting directory
2. *file_name*: The unqualified file name

get_directory()
Returns the directory in which the *FileSet* will be run.

If the directory was set with None in the constructor, get_directory() will return the current working directory.

The returned result is normalized so it never contains a trailing path separator

qualified_files(*absolute=True*)
An alternative generator that yields files rather than directory/file tuples.

If *absolute* is false, paths relative to the starting directory are returned, otherwise files are fully qualified.

class formic.__init__.**Pattern**(*elements*, *casesensitive*)
Bases: object

Represents a single Ant Glob.

The *Pattern* object compiles the pattern into several components:

- *file_pattern*: The a pattern for matching files (not directories) eg, for `test/*.py`, the file_pattern is `*.py`. This is always the text after the final `/` (if any). If the end of the pattern is a `/`, then an implicit `**` is added to the end of the pattern.
- *bound_start*: True if the start of the pattern is 'bound' to the start of the path. If the pattern starts with a `/`, the start is bound.
- *bound_end*: True if the end of the pattern is bound to the immediate parent directory where the file matching is occurring. This is True if the pattern specifies a directory before the file pattern, eg `**/test/*`
- *sections*: A list of `Section` instances. Each `Section` represents a contiguous series of path patterns, and `Section` instances are separated whenever there is a `**` in the glob.

Pattern also normalises the glob, removing redundant path elements (eg `**/**/test/*` resolves to `**/test/*`) and normalises the case of the path elements (resolving difficulties with case insensitive file systems)

all_files()
Returns True if the *Pattern* matches all files (in a matched directory).

The file pattern at the end of the glob was `/` or `/*`

static create(*glob*, *casesensitive=True*)

match_directory(*path_elements*)
Returns a `MatchType` for the directory, expressed as a list of path elements, match for the *Pattern*.

If `self.bound_start` is True, the first `Section` must match from the first directory element.

If `self.bound_end` is True, the last `Section` must match the last contiguous elements of *path_elements*.

match_files(*matched*, *unmatched*)
Moves all matching files from the set *unmatched* to the set *matched*.

Both *matched* and *unmatched* are sets of string, the strings being unqualified file names

formic.__init__.**get_version**()
Returns the version of formic.

This method retrieves the version from VERSION.txt, and it should be exactly the same as the version retrieved from the package manager

exception formic.__init__.**FormicError**(*message=None*)
Bases: exceptions.Exception

Formic errors, such as misconfigured arguments and internal exceptions

5.2 formic Module

An implementation of Ant Globs.

The main entry points for this modules are:

- *FileSet*: A collection of include and exclude globs starting at a specific directory.

  - *FileSet.files()*: A generator returning the matched files as directory/file tuples
  - *FileSet.qualified_files()*: A generator returning the matched files as qualified paths
- *Pattern*: An individual glob

class formic.formic.**ConstantMatcher**(*pattern*, *casesensitive=True*)
Bases: *formic.formic.Matcher*

A *Matcher* for matching the constant passed in the constructor.

This is used to more efficiently match path and file elements that do not have a wild-card, eg `__init__.py`

match(*string*)
Returns True if the argument matches the constant.

class formic.formic.**FNMatcher**(*pattern*, *casesensitive=True*)
Bases: *formic.formic.Matcher*

A *Matcher* that matches simple file/directory wildcards as per DOS or Unix.

- `FNMatcher("*.py")` matches all Python files in a given directory.
- `FNMatcher("?ed")` matches bed, fed, wed but not failed

FNMatcher internally uses `fnmatch.fnmatch()` to implement *Matcher.match()*

match(*string*)
Returns True if the pattern matches the string

class formic.formic.**FileSet**(*include*, *exclude=None*, *directory=None*, *default_excludes=True*, *walk=None*, *symlinks=True*, *casesensitive=True*)
Bases: `object`

An implementation of the Ant FileSet class.

Arguments to the constructor:

1. *include*: An Ant glob or list of Ant globs for matching files to include in the response. Ant globs can be specified either:
   (a) As a string, eg `"*.py"`, or
   (b) As a *Pattern* object
2. *exclude*: Specified in the same was as *include*, but any file that matches an exclude glob will be excluded from the result.
3. *directory*: The directory from which to start the search; if None, the current working directory is used
4. *default_excludes*: A boolean; if True (or omitted) the *DEFAULT_EXCLUDES* will be combined with the *exclude*. If False, the only excludes used are those in the excludes argument
5. *symlinks*: Sets whether symbolic links are included in the results or not. Defaults to True.
6. *walk*: A function whose argument is a single directory that returns a list of (dirname, subdirectoryNames, fileNames) tuples with the same semantics of `os.walk()`. Defaults to `os.walk()`
7. *casesensitive*: Only effective on POSIX, default True. Always False on NT.

Implementation notes:

- *FileSet* is lazy: The files in the *FileSet* are resolved at the time the iterator is looped over. This means that it is very fast to set up and (can be) computationally expensive only when results are obtained.
- You can iterate over the same *FileSet* instance as many times as you want. Because the results are computed as you iterate over the object, each separate iteration can return different results, eg if the file system has changed.

- *include* and *exclude* arguments to the constructor can be given in several ways:
  - A string: This will be automatically turned into a *Pattern*
  - A *Pattern*: If you prefer to construct the pattern yourself
  - A list of strings and/or *Pattern* instances (as above)
- In addition to Apache Ant's default excludes, *FileSet* excludes:
  - __pycache__
- You can modify the *DEFAULT_EXCLUDES* class member (it is a list of *Pattern* instances). Doing so will modify the behaviour of all instances of *FileSet* using default excludes.
- You can provide and alternate function to `os.walk()` that, for example, heavily truncates the files and directories being searched or returns files and directories that don't even exist on the file system. This can be useful for testing or even for passing the results of one FileSet result as the search path of a second. See `formic.treewalk.walk_from_list()`:

```
   files = ["CVS/error.py", "silly/silly1.txt", "1/2/3.py", "silly/silly3.txt
↪", "1/2/4.py", "silly/silly3.txt"]
   fileset = FileSet(include="*.py", walk=treewalk.walk_from_list(files))
   for dir, file in fileset:
       print dir, file

This lists 1/2/3.py and 1/2/4.py no matter what the contents of the
current directory are. CVS/error.py is not listed because of the default
excludes.
```

DEFAULT_EXCLUDES = [/__pycache__/**/*, **/*~, **/#*#, **/.#*, **/%*%, **/._*, **/CVS**
Default excludes shared by all instances. The member is a list of *Pattern* instances. You may modify this member at run time to modify the behaviour of all instances.

files()
A generator function for iterating over the individual files of the FileSet.

The generator yields a tuple of `(rel_dir_name, file_name)`:

1. *rel_dir_name*: The path relative to the starting directory
2. *file_name*: The unqualified file name

get_directory()
Returns the directory in which the *FileSet* will be run.

If the directory was set with None in the constructor, get_directory() will return the current working directory.

The returned result is normalized so it never contains a trailing path separator

qualified_files(*absolute=True*)
An alternative generator that yields files rather than directory/file tuples.

If *absolute* is false, paths relative to the starting directory are returned, otherwise files are fully qualified.

class formic.formic.**FileSetState**(*label*, *directory*, *based_on=None*, *unmatched=None*)
Bases: object

FileSetState is an object encapsulating the *FileSet* in a particular directory, caching inheritable Pattern matches.

This is an internal implementation class and not meant for reuse or to be accessed directly

Implementation notes:

As the FileSet traverses the directories using, by default, `os.walk()`, it builds two graphs of FileSetState instances mirroring the graph of directories - one graph of FileSetState instances is for the **include** globs and the other graph of FileSetState instances for the **exclude**.

FileSetState embodies logic to decide whether to prune whole directories from the search, either by detecting the include patterns cannot match any file within, or by detecting that an exclude matches all files in this directory and sub-directories.

The constructor has the following arguments:

1. *label*: A string used only in the `__str__()` method (for debugging)
2. *directory*: The point in the graph that this FileSetState represents. *directory* is relative to the starting node of the graph
3. *based_on*: A FileSetState from the previous directory traversed by *walk_func()*. This is used as the start point in the graph of FileSetStates to search for the correct parent of this. This is None to create the root node.
4. *unmatched*: Used only when *based_on* is None - the set of initial *Pattern* instances. This is either the original include or exclude globs.

During the construction of the instance, the instance will evaluate the directory patterns in *PatternSet* `self.unmatched` and, for each *Pattern*, perform of of the following actions:

1. If a pattern matches, it will be moved into one of the 'matched' *PatternSet* instances:
    1. `self.matched_inherit`: the directory pattern matches all sub subdirectories as well, eg `/test/**`
    2. `self.matched_and_subdir`: the directory matches this directory and *may* match subdirectories as well, eg `/test/**/more/**`
    3. `self.matched_no_subdir`: the directory matches this directory, **but** cannot match any subdirectory, eg `/test/*`. This pattern will thus not be evaluated in any subdirectory.
2. If the pattern does not match, either:
    (a) It may be valid in subdirectories, so it stays in `self.unmatched`, eg `**/nomatch/*`
    (b) It cannot evaluate to true in any subdirectory, eg `/nomatch/**`. In this case it is removed from all *PatternSet* members in this instance.

match (*files*)
Given a set of files in this directory, returns all the files that match the *Pattern* instances which match this directory.

matches_all_files_all_subdirs()
Returns True if there is a pattern that:

- Matches this directory, and
- Matches all sub-directories, and
- Matches all files (eg ends with "*")

This acts as a terminator for *FileSetState* instances in the excludes graph.

no_possible_matches_in_subdirs()
Returns True if there are no possible matches for any subdirectories of this *FileSetState*.

When this :class:FileSetState is used for an 'include', a return of *True* means we can exclude all subdirectories.

exception formic.formic.**FormicError**(*message=None*)
Bases: exceptions.Exception

Formic errors, such as misconfigured arguments and internal exceptions

class formic.formic.**MatchType**
Bases: object

An enumeration of different match/non-match types to optimize the search algorithm.

There are two special considerations in match results that derive from the fact that Ant globs can be 'bound' to the start of the path being evaluated (eg bound start: `/Documents/**`).

The various match possibilities are bitfields using the members starting `BIT_`.

BIT_ALL_SUBDIRECTORIES = 2

BIT_MATCH = 1

BIT_NO_SUBDIRECTORIES = 4

MATCH = 1

MATCH_ALL_SUBDIRECTORIES = 3

MATCH_BUT_NO_SUBDIRECTORIES = 5

NO_MATCH = 0

NO_MATCH_NO_SUBDIRECTORIES = 4

class formic.formic.**Matcher**(*pattern*, *casesensitive=True*)
Bases: object

An abstract class that holds some pattern to be matched; `matcher.match(string)` returns a boolean indicating whether the string matches the pattern.

The *Matcher.create()* method is a Factory that creates instances of various subclasses.

static create(*pattern*, *casesensitive=True*)
Factory for *Matcher* instances; returns a *Matcher* suitable for matching the supplied pattern

match(*_*)
Matcher is an abstract class - this will raise a *FormicError*

class formic.formic.**Pattern**(*elements*, *casesensitive*)
Bases: object

Represents a single Ant Glob.

The *Pattern* object compiles the pattern into several components:

- *file_pattern*: The a pattern for matching files (not directories) eg, for `test/*.py`, the file_pattern is `*.py`. This is always the text after the final `/` (if any). If the end of the pattern is a `/`, then an implicit `**` is added to the end of the pattern.
- *bound_start*: True if the start of the pattern is 'bound' to the start of the path. If the pattern starts with a `/`, the start is bound.
- *bound_end*: True if the end of the pattern is bound to the immediate parent directory where the file matching is occurring. This is True if the pattern specifies a directory before the file pattern, eg `**/test/*`
- *sections*: A list of *Section* instances. Each *Section* represents a contiguous series of path patterns, and *Section* instances are separated whenever there is a `**` in the glob.

Pattern also normalises the glob, removing redundant path elements (eg `**/**/test/*` resolves to `**/test/*`) and normalises the case of the path elements (resolving difficulties with case insensitive file systems)

all_files()
Returns True if the *Pattern* matches all files (in a matched directory).

The file pattern at the end of the glob was `/` or `/*`

static create(*glob*, *casesensitive=True*)

match_directory(*path_elements*)
Returns a *MatchType* for the directory, expressed as a list of path elements, match for the *Pattern*.

If `self.bound_start` is True, the first *Section* must match from the first directory element.

If `self.bound_end` is True, the last *Section* must match the last contiguous elements of *path_elements*.

match_files(*matched*, *unmatched*)
Moves all matching files from the set *unmatched* to the set *matched*.

Both *matched* and *unmatched* are sets of string, the strings being unqualified file names

class formic.formic.**PatternSet**
Bases: object

A set of *Pattern* instances; *PatternSet* provides a number of operations over the entire set.

PatternSet contains a number of implementation optimizations and is an integral part of various optimizations in *FileSet*.

This class is *not* an implementation of Apache Ant PatternSet

all_files()
Returns True if there is any *Pattern* in the *PatternSet* that matches all files (see *Pattern.all_files()*)

Note that this method is implemented using lazy evaluation so direct access to the member `_all_files` is very likely to result in errors

append(*pattern*)
Adds a *Pattern* to the *PatternSet*

empty()
Returns True if the *PatternSet* is empty

extend(*patterns*)
Extend a *PatternSet* with addition *patterns*

patterns can either be:

- A single *Pattern*
- Another *PatternSet* or
- A list of *Pattern* instances

iter()
An iteration generator that allows the loop to modify the *PatternSet* during the loop

match_files(*matched*, *unmatched*)
Apply the include and exclude filters to those files in *unmatched*, moving those that are included, but not excluded, into the *matched* set.

Both *matched* and *unmatched* are sets of unqualified file names.

remove(*pattern*)
Remove a *Pattern* from the *PatternSet*

class formic.formic.**Section**(*elements*, *casesensitive=True*)
Bases: object

A minimal object that holds fragments of a *Pattern* path.

Each *Section* holds a list of pattern fragments matching some contiguous portion of a full path, separated by /**/ from other *Section* instances.

For example, the *Pattern* /top/second/**/sub/**end/* is stored as a list of three *Section* objects:

1. Section(["top", "second"])
2. Section(["sub"])
3. Section(["end"])

match_iter(*path_elements*, *start_at*)
A generator that searches over *path_elements* (starting from the index *start_at*), yielding for each match.

Each value yielded is the index into *path_elements* to the first element *after* each match. In other words, the returned index has already consumed the matching path elements of this *Section*.

Matches work by finding a contiguous group of path elements that match the list of *Matcher* objects in this *Section* as they are naturally paired.

This method includes an implementation optimization that simplifies the search for *Section* instances containing a single path element. This produces significant performance improvements.

formic.formic.**determine_casesensitive**(*casesensitive*)
Can be True/False on POSIX, but always False on NT.

formic.formic.**get_initial_default_excludes**()
Returns a the default excludes as a list of Patterns.

This will be the initial value of *FileSet.DEFAULT_EXCLUDES*. It is defined in the Ant documentation.

formic.formic.**get_path_components**(*directory*)
Breaks a path to a directory into a (drive, list-of-folders) tuple

Parameters **directory** –

Returns a tuple consisting of the drive (if any) and an ordered list of folder names

formic.formic.**get_version**()
Returns the version of formic.

This method retrieves the version from VERSION.txt, and it should be exactly the same as the version retrieved from the package manager

formic.formic.**is_root**(*directory*)
Returns true if the directory is root (eg / on UNIX or c:on Windows)

formic.formic.**reconstitute_path**(*drive*, *folders*)
Reverts a tuple from *get_path_components* into a path.

Parameters

- **drive** – A drive (eg 'c:'). Only applicable for NT systems
- **folders** – A list of folder names

Returns A path comprising the drive and list of folder names. The path terminate with a *os.path.sep* *only* if it is a root directory

5.3 command Module

The command-line glue-code for **formic**. Call *formic.command.main()* with the command-line arguments.

Full usage of the command is:

```
usage: formic [-i [INCLUDE [INCLUDE ...]]] [-e [EXCLUDE [EXCLUDE ...]]]
              [--no-default-excludes] [--no-symlinks] [--insensitive] [-r] [-h] [--
↪usage]
              [--version]
              [directory]
```

formic.command.**create_parser**()
: Creates and returns the command line parser, an `argparser.ArgumentParser` instance.

formic.command.**entry_point**()
: Entry point for command line; calls *formic.command.main()* and then `sys.exit()` with the return value.

formic.command.**main**(**kw*)
: Command line entry point; arguments must match those defined in in *create_parser()*; returns 0 for success, else 1.

  Example:

  ```
  command.main("-i", "**/*.py", "--no-default-excludes")
  ```

  Runs formic printing out all .py files in the current working directory and its children to `sys.stdout`.

  If *kw* is None, *main()* will use `sys.argv`.

CHAPTER 6

Formic: Better File Fuzzy Searching Tools

6.1 Overview

Formic provides better "file fuzzy searching function" than "find command" on Unix, in my opinion, and it can also work well on Windows.

Formic is forked from https://bitbucket.org/aviser/formic. The original project only supports python2.7 and has not been maintained for a long time.

I added Python3 supports and fixed some issues. Formic now can work on any Python 2.6+ or Python 3.4+ system. If not, please file an issue. Yet not tested on other Python version.

Formic has no runtime dependencies outside the Python system libraries.

6.2 Install

Formic can be installed from the Cheeseshop with easy_install:

```
$ easy_install formic2
```

Or pip:

```
$ pip install formic2
```

6.3 Quickstart

Once installed, you can use Formic either from the command line to find from the current directory:

```
$ formic -i "*.py" -e "__init__.py" "**/*test*/" "test_*"
```

This will search for files all Python files under the current directory excluding all *__init__.py* files, any file in directories whose name contains the word 'test', and any files that start *test_*.

You can also find from the specified directory like below:

```
$ formic /specified/directory/can/ignore/ -i "*.py" "**/test/**/*.txt" "*.ini"
```

Output from Formic is formatted like the Unix find command, and so can easily be combined with other executables, eg:

```
$ formic -i "**/*.bak" | xargs rm
```

will delete all *.bak* files in or under the current directory (but excluding VCS directories such as *.svn* and *.hg*).

Formic can also be integrated right into your Python project:

```
import formic
fileset = formic.FileSet(include="**.py",
                         exclude=["**/*test*/**", "test_*"],
                         directory="./",
                         symlinks=False, )

for file_name in fileset:
    # Do something with file_name
    ...
```

Formic is always case-insensitive on NT, but can be either case-sensitive or case-insensitive on POSIX.

On NT:

```
$ formic ./test/ -i "upp*" "upp*/"
/some/where/formic/test/lower/UPPER.txt
/some/where/formic/test/UPPER/lower.txt
/some/where/formic/test/UPPER/UPPER.txt
```

On POSIX with case-insensitive:

```
$ formic ./test/ --insensitive -i "upp*" "upp*/"
/some/where/formic/test/lower/UPPER.txt
/some/where/formic/test/UPPER/lower.txt
/some/where/formic/test/UPPER/UPPER.txt
```

with case-sensitive:

```
$ formic ./test/ -i "upp*" "upp*/"
$
```

That's about it :)

6.4 Features

Formic is a Python implementation of Apache Ant FileSet and Globs including the directory wildcard **.

FileSet provides a terse way of specifying a set of files without having to enumerate individual files. It:

1. **Includes** files from one or more Ant Globs, then
2. Optionally **excludes** files matching further Ant Globs.

Ant Globs are a superset of ordinary file system globs. The key differences:

- They match whole paths, eg `/root/myapp/*.py`
- ** matches *any* directory or *directories*, eg `/root/**/*.py` matches `/root/one/two/my.py`
- You can match the topmost directory or directories, eg `/root/**`, or
- The parent directory of the file, eg `**/parent/*.py`, or
- Any parent directory, eg `**/test/**/*.py`

This approach is the de-facto standard in several other languages and tools, including Apache Ant and Maven, Ruby (Dir) and Perforce (...).

Python has built-in support for simple globs in fnmatcher and glob, but Formic:

- Can recursively scan subdirectories
- Matches arbitrary directories *in* the path (eg `/1/**/2/**/3/**/*.py`).
- Has a high level interface:
  - Specify one or more globs to find files
  - Globs can be used to exclude files
  - Ant, and Formic, has a set of *default excludes*. These are files and directories that, by default, are automatically excluded from all searches. The majority of these are files and directories related to VCS (eg .svn directories). Formic adds `__pycache__`.
  - Iterate through all matches in the sub-tree
- Is more efficient with many common patterns; it runs relatively faster on large directory trees with large numbers of files.

6.5 About

Formic is originally written and maintained by Andrew Alcock of Aviser LLP, Singapore.

But now, I forked it on GitHub and will maintain this project voluntarily for a long time.

- Origin Homepage
- Current Issue tracker
- Current Source on GitHub
- PyPI
- ReadTheDocs

Python Module Index

f

Index

N

P

Q

R

S