gitmatch

Release 0.1.0

John Thorvald Wodder II

2022 Jul 20

CONTENTS

GitHub | PyPI | Documentation | Issues

CHAPTER

ONE

PATTERNS

The pattern language used by `gitmatch` is intended to match that of Git's gitignore(5) as of v2.36.1, including the undocumented features (mainly involving character classes) present in Git's code.

Specifically:

- A pattern that starts with a `#` or is empty (after stripping trailing whitespace, a trailing `/`, and an initial `!`) is discarded
- Trailing space and tab characters in a pattern are stripped unless they are escaped with a backslash (which must itself not be escaped by another backslash)
- The forward slash (`/`) is used as the directory separator, even on Windows
- An initial `!` negates the pattern; if a path matches a negated pattern, then any matches against previous patterns in the pattern list will be discarded.
- `?` matches any character other than `/`
- `*` matches zero or more of any character other than `/`
- A leading or medial `/` anchors the pattern to the start of the path; if no such `/` is present, the pattern will match any path in which it is preceded by zero or more `/`-separated path components, each one composed of one or more non-`/` characters
- A trailing `/` causes the pattern to only match directories
- An initial `**/` matches zero or more `/`-separated path components
- A trailing `/**` matches one or more `/`-separated path components
- `/**/` matches zero or more intervening `/`-separated path components; e.g., `foo/**/bar` matches `foo/bar`, `foo/gnusto/bar`, `foo/gnusto/cleesh/bar`, etc, but not `fooxbar`. Any following `**/` (e.g., as in `foo/**/**/**/bar`) are redundant.
- A medial `**/` matches zero or more of any character, including `/`
- `**` in any other context is the same as `*`
- `[` starts a character class, which must be terminated by `]`. A character class will match any one character from the set of characters specified within. Characters can be specified as either themselves (e.g., `[abc]` matches `a`, `b`, or `c`) and/or as ranges (e.g., `[a-f]` matches any letter from `a` through `f`).
  - A character class can be inverted (making it match any character except those specified) by inserting `!` or `^` after the opening `[`
  - A `]` can be included in a character set by either escaping it or by placing it immediately after the opening `[` and optional `!/^`.

    - In order for a `]` to be used on the right side of a range, it must be escaped with a backslash; otherwise, it indicates the end of the character class, and the preceding hyphen and character before it will be treated literally rather than as a range.
  - Within a character class, an occurrence of `[:PROPERTY:]` will cause the class to include the ASCII characters with the given property; supported properties are:
    - `alnum` — letters and numbers
    - `alpha` — letters
    - `blank` — space and tab character
    - `cntrl` — any character with an ASCII value less than 0x20, plus the DEL (0x7F) character
    - `digit` — numbers
    - `graph` — letters, numbers, and punctuation
    - `lower` — lowercase letters
    - `print` — letters, numbers, punctuation, and the space character
    - `punct` — punctuation
    - `space` — space character, tab, line feed, and carriage return
    - `upper` — uppercase letters
    - `xdigit` — hexadecimal digits

    An unknown `PROPERTY` produces an invalid pattern that will not match anything.
  - A character class will never match a `/`
- Any character (special or not) in a pattern may be deprived of any special meaning by preceding it with a backslash. A backslash that is not followed by a character (after stripping a final `/`) produces an invalid pattern that will not match anything.
- If a directory path matches a pattern list, then all files & directories within that directory recursively will match as well, regardless of any negative patterns that may apply to them
- Patterns cannot contain the NUL character
- A path containing a NUL character will never match any pattern
- A pattern will never match the current directory

1.1 Strings vs. Bytes

While it's usual in Python to work with `str` values of Unicode characters, Git instead operates on bytes. As a result, if a path or pattern contains non-ASCII characters, you may get different results using `str`s with `gitmatch` than you would with Git. For example, in Git, a file named "`tést`" will not be matched by the gitignore pattern `t?st`, because the `é` is encoded using more than one byte (assuming UTF-8), but if you pass these strings to `gitmatch`, the path will match (assuming the `é` is in composed form, which is a whole other can of worms). If you want Git's behavior exactly, pass `bytes` to `gitmatch` instead of `str` (ideally encoded using `os.fsencode()`).

Note that the patterns passed to a single call to *`gitmatch.compile()`* must be either all `str` or all `bytes`, and a *`Gitignore`* instance constructed from `str` patterns can only match against `str` paths, while one constructed from `bytes` patterns can only match against `bytes` paths. (For the record, the `pathlib` classes count as `str` paths.)

CHAPTER

TWO

API

2.1 Functions

gitmatch.**compile**(*patterns: Iterable*, *ignorecase: bool = False*) → *Gitignore*

Compile a collection of gitignore patterns into a *Gitignore* instance. Any invalid or empty patterns are discarded.

Trailing newlines are stripped from the patterns before compiling, so you can compile a pre-existing `.gitignore` file by simply doing:

```
with open("path/to/.gitignore") as fp:
    gi = gitmatch.compile(fp)
```

Parameters

- **patterns** – an iterable of gitignore patterns
- **ignorecase** (*bool*) – Whether the patterns should match case-insensitively

gitmatch.**pattern2regex**(*pattern: AnyStr*, *ignorecase: bool = False*) → Optional[*Regex*]

Convert a gitignore pattern to a regular expression and return a *Regex* object. If the pattern is empty or a comment, returns `None`.

Parameters

- **pattern** – a gitignore pattern
- **ignorecase** (*bool*) – Whether the pattern should match case-insensitively

Raises

InvalidPatternError – If the given pattern is invalid

2.2 Classes

Note: Although the Sphinx docs don't show it, all of the `gitmatch` classes are generic in `typing.AnyStr`; i.e., they should be written in type annotations as `Gitignore[AnyStr]`, `Gitignore[str]`, or `Gitignore[bytes]`, as appropriate.

class gitmatch.**Gitignore**

A collection of compiled gitignore patterns

match(*path: Union[AnyStr, PathLike]*, *is_dir: bool = False*) → Optional[*Match*]

Test whether the given relative path matches the collection of patterns. If `is_dir` is true or if `path` ends in a slash, `path` is treated as a path to a directory; otherwise, it treated as a path to a file.

If on Windows and `path` is not an instance of `pathlib.PurePosixPath`, or if on any OS and `path` is an instance of `pathlib.PureWindowsPath`, any backslashes in `path` will be converted to forward slashes before matching.

If a match is found, a *Match* object is returned containing information about the matching pattern and the path or portion thereof that matched. The *Match* object may be either "truthy" or "falsy" depending on whether the matching pattern is negative or not. If none of the patterns match the path, *match()* returns `None`. Hence, if you're just interested in whether the patterns say the path should be gitignored, call `bool()` on the result or use it in a boolean context like an `if ... :` line.

Raises

InvalidPathError – If `path` is empty, is absolute, is not normalized (aside from an optional trailing slash), contains a NUL character, or starts with `...`

class gitmatch.**Pattern**

A compiled gitignore pattern

dir_only: bool

Whether the pattern only matches directories

ignorecase: bool

Whether the pattern is case-insensitive

match(*path: AnyStr*, *is_dir: bool = False*) → bool

Test whether the pattern matches the given path. `path` is assumed to be a relative, normalized, `/`-separated path. If `is_dir` is true, the path is assumed to refer to a directory; otherwise, it is assumed to refer to a file.

Unlike *Gitignore.match()*, this method only tests `path` itself, not any of its parent paths.

negative: bool

Whether the pattern is negative or not

pattern: AnyStr

The original gitignore pattern provided to *compile()*, with trailing spaces stripped

regex: Pattern

A compiled regular expression pattern

class gitmatch.**Regex**

A gitignore pattern that has been converted to a regular expression

compile() → *Pattern*

Compile the regular expression

dir_only: bool

Whether the pattern only matches directories

ignorecase: bool

Whether the pattern is case-insensitive

negative: bool

Whether the pattern is negative or not

pattern: AnyStr

The original gitignore pattern provided to *compile()*, with trailing spaces stripped

regex: AnyStr

The regular expression equivalent of the pattern

class gitmatch.**Match**

Information about a successful match of a path against a pattern. A *Match* is truthy if the pattern was not negative and falsy otherwise.

path: AnyStr

The path that matched. This may be a parent path of the value passed to *match()*.

property pattern: AnyStr

The original gitignore pattern provided to *compile()*, with trailing spaces stripped

pattern_obj: *Pattern*

The compiled *Pattern* object that matched the path

2.3 Exceptions

exception gitmatch.**InvalidPathError**

Bases: ValueError

Raised by *Gitignore.match()* when given an invalid path

msg

A description of the problem with the path

path

The invalid path

exception gitmatch.**InvalidPatternError**

Bases: ValueError

Raised by *pattern2regex()* when given an invalid pattern

pattern

The invalid pattern

gitmatch provides gitignore-style pattern matching of file paths. Simply pass in a sequence of gitignore patterns and you'll get back an object for testing whether a given relative path matches the patterns.

CHAPTER

THREE

INSTALLATION

`gitmatch` requires Python 3.7 or higher. Just use pip for Python 3 (You have pip, right?) to install it:

```
python3 -m pip install gitmatch
```

CHAPTER

FOUR

EXAMPLES

Basic usage:

```
>>> import gitmatch
>>> gi = gitmatch.compile(["foo", "!bar", "*.dir/"])
>>> bool(gi.match("foo"))
True
>>> bool(gi.match("bar"))
False
>>> bool(gi.match("quux"))
False
>>> bool(gi.match("foo/quux"))
True
>>> bool(gi.match("foo/bar"))
True
>>> bool(gi.match("bar/foo"))
True
>>> bool(gi.match("bar/quux"))
False
>>> bool(gi.match("foo.dir"))
False
>>> bool(gi.match("foo.dir/"))
True
```

See what pattern was matched:

```
>>> m1 = gi.match("foo/bar")
>>> m1 is None
False
>>> bool(m1)
True
>>> m1.pattern
'foo'
>>> m1.path
'foo'
>>> m2 = gi.match("bar")
>>> m2 is None
False
>>> bool(m2)
False
>>> m2.pattern
'!bar'
```

(continues on next page)

(continued from previous page)

```
>>> m2.pattern_obj.negative
True
>>> m3 = gi.match("quux")
>>> m3 is None
True
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

INDEX