minimatch

A minimal matching utility.

Build Status

This is the matching library used internally by npm.

It works by converting glob expressions into JavaScript `RegExp` objects.

Usage

```
var minimatch = require("minimatch")

minimatch("bar.foo", "*.foo") // true!
minimatch("bar.foo", "*.bar") // false!
minimatch("bar.foo", "*.+(bar|foo)", {
        debug: true }) // true, and
        noisy!
```

Features

Supports these glob features:

- Brace Expansion
- Extended glob matching
- “Globstar” ** matching

See:

- `man sh`
- `man bash`
- `man 3 fnmatch`
- `man 5 gitignore`

Minimatch Class

Create a minimatch object by instantiating the `minimatch.Minimatch` class.

```
var Minimatch =
        require("minimatch").Minimatch
var mm = new Minimatch(pattern, options)
```

Properties

- `pattern` The original pattern the minimatch object represents.
- `options` The options supplied to the constructor.
- `set` A 2-dimensional array of regexp or string expressions. Each row in the array corresponds to a brace-expanded pattern. Each item in the row corresponds to a single path-part. For example, the pattern `{a,b/c}/d` would expand to a set of patterns like:

If brace expansion is not disabled, then it is performed before any other interpretation of the glob pattern. Thus, a pattern like `+(a|{b),c)}`, which would not be valid in bash or zsh, is expanded **first** into the set of `+(a|b)` and `+(a|c)`, and those patterns are checked for validity. Since those two are valid, matching proceeds.


```
    [ [ a, d ]
  , [ b, c, d ] ]
```

If a portion of the pattern doesn't have any "magic" in it (that is, it's something like `"foo"` rather than `fo*o?`), then it will be left as a string rather than converted to a regular expression.

- `regexp` Created by the `makeRe` method. A single regular expression expressing the entire pattern. This is useful in cases where you wish to use the pattern somewhat like `fnmatch(3)` with `FNM_PATH` enabled.
- `negate` True if the pattern is negated.
- `comment` True if the pattern is a comment.
- `empty` True if the pattern is `""`.

Methods

- `makeRe` Generate the `regexp` member if necessary, and return it. Will return `false` if the pattern is invalid.
- `match(fname)` Return true if the filename matches the pattern, or false otherwise.
- `matchOne(fileArray, patternArray, partial)` Take a `/`-split filename, and match it against a single row in the `regExpSet`. This method is mainly for internal use, but is exposed so that it can be used by a glob-walker that needs to avoid excessive filesystem calls.

All other methods are internal, and will be called as necessary.

minimatch(path, pattern, options)

Main export. Tests a path against the pattern using the options.

```
var isJS = minimatch(file, "*.js", {
        matchBase: true })
```

minimatch.filter(pattern, options)

Returns a function that tests its supplied argument, suitable for use with `Array.filter`. Example:

```
var javascripts =
        fileList.filter(minimatch.filter("*.js",
        {matchBase: true}))
```

minimatch.match(list, pattern, options)

Match against the list of files, in the style of fnmatch or glob. If nothing is matched, and options.nonull is set, then return a list containing the pattern itself.

```
var javascripts =
        minimatch.match(fileList,
        "*.js", {matchBase: true}))
```


Comparisons to other fnmatch/glob implementations

While strict compliance with the existing standards is a worthwhile goal, some discrepancies exist between minimatch and other implementations, and are intentional.

If the pattern starts with a `!` character, then it is negated. Set the `nonegate` flag to suppress this behavior, and treat leading `!` characters normally. This is perhaps relevant if you wish to start the pattern with a negative extglob pattern like `!(a|B)`. Multiple `!` characters at the start of a pattern will negate the pattern multiple times.

If a pattern starts with #, then it is treated as a comment, and will not match anything. Use `\#` to match a literal # at the start of a line, or set the `nocomment` flag to suppress this behavior.

The double-star character `**` is supported by default, unless the `noglobstar` flag is set. This is supported in the manner of bsdglob and bash 4.1, where `**` only has special significance if it is the only thing in a path part. That is, `a/**/b` will match `a/x/y/b`, but `a/**b` will not.

If an escaped pattern has no matches, and the `nonull` flag is set, then minimatch.match returns the pattern as-provided, rather than interpreting the character escapes. For example, `minimatch.match([], "\\*a\\?")` will return `"\\*a\\?"` rather than `"*a?"`. This is akin to setting the `nullglob` option in bash, except that it does not resolve escaped pattern characters.

partial

Compare a partial path to a pattern. As long as the parts of the path that are present are not contradicted by the pattern, it will be treated as a match. This is useful in applications where you're walking through a folder structure, and don't yet have the full path, but want to ensure that you do not walk down paths that can never be a match.

For example,

```
minimatch('/a/b', '/a/*/c/d', {
        partial:
        true })  // true, might be /a/b/
        c/d
minimatch('/a/b', '/**/d', { partial:
        true })     // true, might be /
        a/b/.../d
minimatch('/x/y/z', '/a/**/z', {
        partial: true }) // false,
        because x !== a
```

allowWindowsEscape

Windows path separator `\` is by default converted to `/`, which prohibits the usage of `\` as a escape character. This flag skips that behavior and allows using the escape character.

minimatch.makeRe(pattern, options)

Make a regular expression object from the pattern.

Options

All options are `false` by default.

debug

Dump a ton of stuff to stderr.

nobrace

Do not expand `{a,b}` and `{1..3}` brace sets.

noglobstar

Disable `**` matching against multiple folder names.

dot

Allow patterns to match filenames starting with a period, even if the pattern does not explicitly have a period in that spot.

Note that by default, `a/**/b` will **not** match `a/.d/b`, unless `dot` is set.

noext

Disable “extglob” style patterns like `+(a|b)`.

nocase

Perform a case-insensitive match.

nonull

When a match is not found by `minimatch.match`, return a list containing the pattern itself if this option is set. When not set, an empty list is returned if there are no matches.

matchBase

If set, then patterns without slashes will be matched against the basename of the path if it contains slashes. For example, `a?b` would match the path `/xyz/123/acb`, but not `/xyz/acb/123`.

nocomment

Suppress the behavior of treating `#` at the start of a pattern as a comment.

nonegate

Suppress the behavior of treating a leading `!` character as negation.

flipNegate

Returns from negate expressions the same as if they were not negated. (Ie, true on a hit, false on a miss.)