# Glob

Match files using the patterns the shell uses.

The most correct and second fastest glob implementation in JavaScript. (See **Comparison to Other JavaScript Glob Implementations** at the bottom of this readme.)



## Usage

Install with npm

npm i glob

**Note** the npm package name is *not* node-glob that's a different thing that was abandoned years ago. Just glob.

// load using import

import { glob, globSync, globStream, globStreamSync, Glob } from 'glob'

// or using commonjs, that's fine, too

const {

glob,

globSync,

globStream,

globStreamSync,

Glob,

} = require('glob')

// the main glob() and globSync() resolve/return array of filenames

// all js files, but don't look in node\_modules

const jsfiles = await glob('\*\*/\*.js', { ignore: 'node\_modules/\*\*' })

// pass in a signal to cancel the glob walk

const stopAfter100ms = await glob('\*\*/\*.css', {

signal: AbortSignal.timeout(100),

})

// multiple patterns supported as well

const images = await glob(['css/\*.{png,jpeg}', 'public/\*.{png,jpeg}'])

// but of course you can do that with the glob pattern also

// the sync function is the same, just returns a string[] instead

// of Promise<string[]>

const imagesAlt = globSync('{css,public}/\*.{png,jpeg}')

// you can also stream them, this is a Minipass stream

const filesStream = globStream(['\*\*/\*.dat', 'logs/\*\*/\*.log'])

// construct a Glob object if you wanna do it that way, which

// allows for much faster walks if you have to look in the same

// folder multiple times.

const g = new Glob('\*\*/foo', {})

// glob objects are async iterators, can also do globIterate() or

// g.iterate(), same deal

for await (const file of g) {

console.log('found a foo file:', file)

}

// pass a glob as the glob options to reuse its settings and caches

const g2 = new Glob('\*\*/bar', g)

// sync iteration works as well

for (const file of g2) {

console.log('found a bar file:', file)

}

// you can also pass withFileTypes: true to get Path objects

// these are like a Dirent, but with some more added powers

// check out http://npm.im/path-scurry for more info on their API

const g3 = new Glob('\*\*/baz/\*\*', { withFileTypes: true })

g3.stream().on('data', path => {

console.log(

'got a path object',

path.fullpath(),

path.isDirectory(),

path.readdirSync().map(e => e.name),

)

})

// if you use stat:true and withFileTypes, you can sort results

// by things like modified time, filter by permission mode, etc.

// All Stats fields will be available in that case. Slightly

// slower, though.

// For example:

const results = await glob('\*\*', { stat: true, withFileTypes: true })

const timeSortedFiles = results

.sort((a, b) => a.mtimeMs - b.mtimeMs)

.map(path => path.fullpath())

const groupReadableFiles = results

.filter(path => path.mode & 0o040)

.map(path => path.fullpath())

// custom ignores can be done like this, for example by saying

// you'll ignore all markdown files, and all folders named 'docs'

const customIgnoreResults = await glob('\*\*', {

ignore: {

ignored: p => /\.md$/.test(p.name),

childrenIgnored: p => p.isNamed('docs'),

},

})

// another fun use case, only return files with the same name as

// their parent folder, plus either `.ts` or `.js`

const folderNamedModules = await glob('\*\*/\*.{ts,js}', {

ignore: {

ignored: p => {

const pp = p.parent

return !(p.isNamed(pp.name + '.ts') || p.isNamed(pp.name + '.js'))

},

},

})

// find all files edited in the last hour, to do this, we ignore

// all of them that are more than an hour old

const newFiles = await glob('\*\*', {

// need stat so we have mtime

stat: true,

// only want the files, not the dirs

nodir: true,

ignore: {

ignored: p => {

return new Date() - p.mtime > 60 \* 60 \* 1000

},

// could add similar childrenIgnored here as well, but

// directory mtime is inconsistent across platforms, so

// probably better not to, unless you know the system

// tracks this reliably.

},

})

**Note** Glob patterns should always use / as a path separator, even on Windows systems, as \ is used to escape glob characters. If you wish to use \ as a path separator *instead of* using it as an escape character on Windows platforms, you may set windowsPathsNoEscape:true in the options. In this mode, special glob characters cannot be escaped, making it impossible to match a literal \* ? and so on in filenames.

## Command Line Interface

$ glob -h

Usage:

glob [options] [<pattern> [<pattern> ...]]

Expand the positional glob expression arguments into any matching file system

paths found.

-c<command> --cmd=<command>

Run the command provided, passing the glob expression

matches as arguments.

-A --all By default, the glob cli command will not expand any

arguments that are an exact match to a file on disk.

This prevents double-expanding, in case the shell

expands an argument whose filename is a glob

expression.

For example, if 'app/\*.ts' would match 'app/[id].ts',

then on Windows powershell or cmd.exe, 'glob app/\*.ts'

will expand to 'app/[id].ts', as expected. However, in

posix shells such as bash or zsh, the shell will first

expand 'app/\*.ts' to a list of filenames. Then glob

will look for a file matching 'app/[id].ts' (ie,

'app/i.ts' or 'app/d.ts'), which is unexpected.

Setting '--all' prevents this behavior, causing glob to

treat ALL patterns as glob expressions to be expanded,

even if they are an exact match to a file on disk.

When setting this option, be sure to enquote arguments

so that the shell will not expand them prior to passing

them to the glob command process.

-a --absolute Expand to absolute paths

-d --dot-relative Prepend './' on relative matches

-m --mark Append a / on any directories matched

-x --posix Always resolve to posix style paths, using '/' as the

directory separator, even on Windows. Drive letter

absolute matches on Windows will be expanded to their

full resolved UNC maths, eg instead of 'C:\foo\bar', it

will expand to '//?/C:/foo/bar'.

-f --follow Follow symlinked directories when expanding '\*\*'

-R --realpath Call 'fs.realpath' on all of the results. In the case

of an entry that cannot be resolved, the entry is

omitted. This incurs a slight performance penalty, of

course, because of the added system calls.

-s --stat Call 'fs.lstat' on all entries, whether required or not

to determine if it's a valid match.

-b --match-base Perform a basename-only match if the pattern does not

contain any slash characters. That is, '\*.js' would be

treated as equivalent to '\*\*/\*.js', matching js files

in all directories.

--dot Allow patterns to match files/directories that start

with '.', even if the pattern does not start with '.'

--nobrace Do not expand {...} patterns

--nocase Perform a case-insensitive match. This defaults to

'true' on macOS and Windows platforms, and false on all

others.

Note: 'nocase' should only be explicitly set when it is

known that the filesystem's case sensitivity differs

from the platform default. If set 'true' on

case-insensitive file systems, then the walk may return

more or less results than expected.

--nodir Do not match directories, only files.

Note: to \*only\* match directories, append a '/' at the

end of the pattern.

--noext Do not expand extglob patterns, such as '+(a|b)'

--noglobstar Do not expand '\*\*' against multiple path portions. Ie,

treat it as a normal '\*' instead.

--windows-path-no-escape

Use '\' as a path separator \*only\*, and \*never\* as an

escape character. If set, all '\' characters are

replaced with '/' in the pattern.

-D<n> --max-depth=<n> Maximum depth to traverse from the current working

directory

-C<cwd> --cwd=<cwd> Current working directory to execute/match in

-r<root> --root=<root> A string path resolved against the 'cwd', which is used

as the starting point for absolute patterns that start

with '/' (but not drive letters or UNC paths on

Windows).

Note that this \*doesn't\* necessarily limit the walk to

the 'root' directory, and doesn't affect the cwd

starting point for non-absolute patterns. A pattern

containing '..' will still be able to traverse out of

the root directory, if it is not an actual root

directory on the filesystem, and any non-absolute

patterns will still be matched in the 'cwd'.

To start absolute and non-absolute patterns in the same

path, you can use '--root=' to set it to the empty

string. However, be aware that on Windows systems, a

pattern like 'x:/\*' or '//host/share/\*' will \*always\*

start in the 'x:/' or '//host/share/' directory,

regardless of the --root setting.

--platform=<platform> Defaults to the value of 'process.platform' if

available, or 'linux' if not. Setting --platform=win32

on non-Windows systems may cause strange behavior!

-i<ignore> --ignore=<ignore>

Glob patterns to ignore Can be set multiple times

-v --debug Output a huge amount of noisy debug information about

patterns as they are parsed and used to match files.

-h --help Show this usage information

## glob(pattern: string | string[], options?: GlobOptions) => Promise<string[] | Path[]>

Perform an asynchronous glob search for the pattern(s) specified. Returns [Path](https://isaacs.github.io/path-scurry/classes/PathBase) objects if the withFileTypes option is set to true. See below for full options field desciptions.

## globSync(pattern: string | string[], options?: GlobOptions) => string[] | Path[]

Synchronous form of glob().

Alias: glob.sync()

## globIterate(pattern: string | string[], options?: GlobOptions) => AsyncGenerator<string>

Return an async iterator for walking glob pattern matches.

Alias: glob.iterate()

## globIterateSync(pattern: string | string[], options?: GlobOptions) => Generator<string>

Return a sync iterator for walking glob pattern matches.

Alias: glob.iterate.sync(), glob.sync.iterate()

## globStream(pattern: string | string[], options?: GlobOptions) => Minipass<string | Path>

Return a stream that emits all the strings or Path objects and then emits end when completed.

Alias: glob.stream()

## globStreamSync(pattern: string | string[], options?: GlobOptions) => Minipass<string | Path>

Syncronous form of globStream(). Will read all the matches as fast as you consume them, even all in a single tick if you consume them immediately, but will still respond to backpressure if they're not consumed immediately.

Alias: glob.stream.sync(), glob.sync.stream()

## hasMagic(pattern: string | string[], options?: GlobOptions) => boolean

Returns true if the provided pattern contains any "magic" glob characters, given the options provided.

Brace expansion is not considered "magic" unless the magicalBraces option is set, as brace expansion just turns one string into an array of strings. So a pattern like 'x{a,b}y' would return false, because 'xay' and 'xby' both do not contain any magic glob characters, and it's treated the same as if you had called it on ['xay', 'xby']. When magicalBraces:true is in the options, brace expansion *is* treated as a pattern having magic.

## escape(pattern: string, options?: GlobOptions) => string

Escape all magic characters in a glob pattern, so that it will only ever match literal strings

If the windowsPathsNoEscape option is used, then characters are escaped by wrapping in [], because a magic character wrapped in a character class can only be satisfied by that exact character.

Slashes (and backslashes in windowsPathsNoEscape mode) cannot be escaped or unescaped.

## unescape(pattern: string, options?: GlobOptions) => string

Un-escape a glob string that may contain some escaped characters.

If the windowsPathsNoEscape option is used, then square-brace escapes are removed, but not backslash escapes. For example, it will turn the string '[\*]' into \*, but it will not turn '\\\*' into '\*', because \ is a path separator in windowsPathsNoEscape mode.

When windowsPathsNoEscape is not set, then both brace escapes and backslash escapes are removed.

Slashes (and backslashes in windowsPathsNoEscape mode) cannot be escaped or unescaped.

## Class Glob

An object that can perform glob pattern traversals.

### const g = new Glob(pattern: string | string[], options: GlobOptions)

Options object is required.

See full options descriptions below.

Note that a previous Glob object can be passed as the GlobOptions to another Glob instantiation to re-use settings and caches with a new pattern.

Traversal functions can be called multiple times to run the walk again.

### g.stream()

Stream results asynchronously,

### g.streamSync()

Stream results synchronously.

### g.iterate()

Default async iteration function. Returns an AsyncGenerator that iterates over the results.

### g.iterateSync()

Default sync iteration function. Returns a Generator that iterates over the results.

### g.walk()

Returns a Promise that resolves to the results array.

### g.walkSync()

Returns a results array.

### Properties

All options are stored as properties on the Glob object.

* opts The options provided to the constructor.
* patterns An array of parsed immutable Pattern objects.

## Options

Exported as GlobOptions TypeScript interface. A GlobOptions object may be provided to any of the exported methods, and must be provided to the Glob constructor.

All options are optional, boolean, and false by default, unless otherwise noted.

All resolved options are added to the Glob object as properties.

If you are running many glob operations, you can pass a Glob object as the options argument to a subsequent operation to share the previously loaded cache.

* cwd String path or file:// string or URL object. The current working directory in which to search. Defaults to process.cwd(). See also: "Windows, CWDs, Drive Letters, and UNC Paths", below.

This option may be either a string path or a file:// URL object or string.

* root A string path resolved against the cwd option, which is used as the starting point for absolute patterns that start with /, (but not drive letters or UNC paths on Windows).

Note that this *doesn't* necessarily limit the walk to the root directory, and doesn't affect the cwd starting point for non-absolute patterns. A pattern containing .. will still be able to traverse out of the root directory, if it is not an actual root directory on the filesystem, and any non-absolute patterns will be matched in the cwd. For example, the pattern /../\* with {root:'/some/path'} will return all files in /some, not all files in /some/path. The pattern \* with {root:'/some/path'} will return all the entries in the cwd, not the entries in /some/path.

To start absolute and non-absolute patterns in the same path, you can use {root:''}. However, be aware that on Windows systems, a pattern like x:/\* or //host/share/\* will *always* start in the x:/ or //host/share directory, regardless of the root setting.

* windowsPathsNoEscape Use \\ as a path separator *only*, and *never* as an escape character. If set, all \\ characters are replaced with / in the pattern.

Note that this makes it **impossible** to match against paths containing literal glob pattern characters, but allows matching with patterns constructed using path.join() and path.resolve() on Windows platforms, mimicking the (buggy!) behavior of Glob v7 and before on Windows. Please use with caution, and be mindful of [the caveat below about Windows paths](#windows). (For legacy reasons, this is also set if allowWindowsEscape is set to the exact value false.)

* dot Include .dot files in normal matches and globstar matches. Note that an explicit dot in a portion of the pattern will always match dot files.
* magicalBraces Treat brace expansion like {a,b} as a "magic" pattern. Has no effect if {@link nobrace} is set.

Only has effect on the {@link hasMagic} function, no effect on glob pattern matching itself.

* dotRelative Prepend all relative path strings with ./ (or .\ on Windows).

Without this option, returned relative paths are "bare", so instead of returning './foo/bar', they are returned as 'foo/bar'.

Relative patterns starting with '../' are not prepended with ./, even if this option is set.

* mark Add a / character to directory matches. Note that this requires additional stat calls.
* nobrace Do not expand {a,b} and {1..3} brace sets.
* noglobstar Do not match \*\* against multiple filenames. (Ie, treat it as a normal \* instead.)
* noext Do not match "extglob" patterns such as +(a|b).
* nocase Perform a case-insensitive match. This defaults to true on macOS and Windows systems, and false on all others.

**Note** nocase should only be explicitly set when it is known that the filesystem's case sensitivity differs from the platform default. If set true on case-sensitive file systems, or false on case-insensitive file systems, then the walk may return more or less results than expected.

* maxDepth Specify a number to limit the depth of the directory traversal to this many levels below the cwd.
* matchBase Perform a basename-only match if the pattern does not contain any slash characters. That is, \*.js would be treated as equivalent to \*\*/\*.js, matching all js files in all directories.
* nodir Do not match directories, only files. (Note: to match *only* directories, put a / at the end of the pattern.)

Note: when follow and nodir are both set, then symbolic links to directories are also omitted.

* stat Call lstat() on all entries, whether required or not to determine whether it's a valid match. When used with withFileTypes, this means that matches will include data such as modified time, permissions, and so on. Note that this will incur a performance cost due to the added system calls.
* ignore string or string[], or an object with ignore and ignoreChildren methods.

If a string or string[] is provided, then this is treated as a glob pattern or array of glob patterns to exclude from matches. To ignore all children within a directory, as well as the entry itself, append '/\*\*' to the ignore pattern.

**Note** ignore patterns are *always* in dot:true mode, regardless of any other settings.

If an object is provided that has ignored(path) and/or childrenIgnored(path) methods, then these methods will be called to determine whether any Path is a match or if its children should be traversed, respectively.

* follow Follow symlinked directories when expanding \*\* patterns. This can result in a lot of duplicate references in the presence of cyclic links, and make performance quite bad.

By default, a \*\* in a pattern will follow 1 symbolic link if it is not the first item in the pattern, or none if it is the first item in the pattern, following the same behavior as Bash.

Note: when follow and nodir are both set, then symbolic links to directories are also omitted.

* realpath Set to true to call fs.realpath on all of the results. In the case of an entry that cannot be resolved, the entry is omitted. This incurs a slight performance penalty, of course, because of the added system calls.
* absolute Set to true to always receive absolute paths for matched files. Set to false to always receive relative paths for matched files.

By default, when this option is not set, absolute paths are returned for patterns that are absolute, and otherwise paths are returned that are relative to the cwd setting.

This does *not* make an extra system call to get the realpath, it only does string path resolution.

absolute may not be used along with withFileTypes.

* posix Set to true to use / as the path separator in returned results. On posix systems, this has no effect. On Windows systems, this will return / delimited path results, and absolute paths will be returned in their full resolved UNC path form, eg insted of 'C:\\foo\\bar', it will return //?/C:/foo/bar.
* platform Defaults to value of process.platform if available, or 'linux' if not. Setting platform:'win32' on non-Windows systems may cause strange behavior.
* withFileTypes Return [PathScurry](http://npm.im/path-scurry) Path objects instead of strings. These are similar to a NodeJS Dirent object, but with additional methods and properties.

withFileTypes may not be used along with absolute.

* signal An AbortSignal which will cancel the Glob walk when triggered.
* fs An override object to pass in custom filesystem methods. See [PathScurry docs](http://npm.im/path-scurry) for what can be overridden.
* scurry A [PathScurry](http://npm.im/path-scurry) object used to traverse the file system. If the nocase option is set explicitly, then any provided scurry object must match this setting.
* includeChildMatches boolean, default true. Do not match any children of any matches. For example, the pattern \*\*\/foo would match a/foo, but not a/foo/b/foo in this mode.

This is especially useful for cases like "find all node\_modules folders, but not the ones in node\_modules".

In order to support this, the Ignore implementation must support an add(pattern: string) method. If using the default Ignore class, then this is fine, but if this is set to false, and a custom Ignore is provided that does not have an add() method, then it will throw an error.

**Caveat** It *only* ignores matches that would be a descendant of a previous match, and only if that descendant is matched *after* the ancestor is encountered. Since the file system walk happens in indeterminate order, it's possible that a match will already be added before its ancestor, if multiple or braced patterns are used.

For example:

const results = await glob(

[

// likely to match first, since it's just a stat

'a/b/c/d/e/f',

// this pattern is more complicated! It must to various readdir()

// calls and test the results against a regular expression, and that

// is certainly going to take a little bit longer.

//

// So, later on, it encounters a match at 'a/b/c/d/e', but it's too

// late to ignore a/b/c/d/e/f, because it's already been emitted.

'a/[bdf]/?/[a-z]/\*',

],

{ includeChildMatches: false },

)

It's best to only set this to false if you can be reasonably sure that no components of the pattern will potentially match one another's file system descendants, or if the occasional included child entry will not cause problems.

## Glob Primer

Much more information about glob pattern expansion can be found by running man bash and searching for Pattern Matching.

"Globs" are the patterns you type when you do stuff like ls \*.js on the command line, or put build/\* in a .gitignore file.

Before parsing the path part patterns, braced sections are expanded into a set. Braced sections start with { and end with }, with 2 or more comma-delimited sections within. Braced sections may contain slash characters, so a{/b/c,bcd} would expand into a/b/c and abcd.

The following characters have special magic meaning when used in a path portion. With the exception of \*\*, none of these match path separators (ie, / on all platforms, and \ on Windows).

* \* Matches 0 or more characters in a single path portion. When alone in a path portion, it must match at least 1 character. If dot:true is not specified, then \* will not match against a . character at the start of a path portion.
* ? Matches 1 character. If dot:true is not specified, then ? will not match against a . character at the start of a path portion.
* [...] Matches a range of characters, similar to a RegExp range. If the first character of the range is ! or ^ then it matches any character not in the range. If the first character is ], then it will be considered the same as \], rather than the end of the character class.
* !(pattern|pattern|pattern) Matches anything that does not match any of the patterns provided. May *not* contain / characters. Similar to \*, if alone in a path portion, then the path portion must have at least one character.
* ?(pattern|pattern|pattern) Matches zero or one occurrence of the patterns provided. May *not* contain / characters.
* +(pattern|pattern|pattern) Matches one or more occurrences of the patterns provided. May *not* contain / characters.
* \*(a|b|c) Matches zero or more occurrences of the patterns provided. May *not* contain / characters.
* @(pattern|pat\*|pat?erN) Matches exactly one of the patterns provided. May *not* contain / characters.
* \*\* If a "globstar" is alone in a path portion, then it matches zero or more directories and subdirectories searching for matches. It does not crawl symlinked directories, unless {follow:true} is passed in the options object. A pattern like a/b/\*\* will only match a/b if it is a directory. Follows 1 symbolic link if not the first item in the pattern, or 0 if it is the first item, unless follow:true is set, in which case it follows all symbolic links.

[:class:] patterns are supported by this implementation, but [=c=] and [.symbol.] style class patterns are not.

### Dots

If a file or directory path portion has a . as the first character, then it will not match any glob pattern unless that pattern's corresponding path part also has a . as its first character.

For example, the pattern a/.\*/c would match the file at a/.b/c. However the pattern a/\*/c would not, because \* does not start with a dot character.

You can make glob treat dots as normal characters by setting dot:true in the options.

### Basename Matching

If you set matchBase:true in the options, and the pattern has no slashes in it, then it will seek for any file anywhere in the tree with a matching basename. For example, \*.js would match test/simple/basic.js.

### Empty Sets

If no matching files are found, then an empty array is returned. This differs from the shell, where the pattern itself is returned. For example:

$ echo a\*s\*d\*f

a\*s\*d\*f

## Comparisons to other fnmatch/glob implementations

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

The double-star character \*\* is supported by default, unless the noglobstar flag is set. This is supported in the manner of bsdglob and bash 5, 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.

Note that symlinked directories are not traversed as part of a \*\*, though their contents may match against subsequent portions of the pattern. This prevents infinite loops and duplicates and the like. You can force glob to traverse symlinks with \*\* by setting {follow:true} in the options.

There is no equivalent of the nonull option. A pattern that does not find any matches simply resolves to nothing. (An empty array, immediately ended stream, etc.)

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.

The character class patterns [:class:] (posix standard named classes) style class patterns are supported and unicode-aware, but [=c=] (locale-specific character collation weight), and [.symbol.] (collating symbol), are not.

### Repeated Slashes

Unlike Bash and zsh, repeated / are always coalesced into a single path separator.

### Comments and Negation

Previously, this module let you mark a pattern as a "comment" if it started with a # character, or a "negated" pattern if it started with a ! character.

These options were deprecated in version 5, and removed in version 6.

To specify things that should not match, use the ignore option.

## Windows

**Please only use forward-slashes in glob expressions.**

Though windows uses either / or \ as its path separator, only / characters are used by this glob implementation. You must use forward-slashes **only** in glob expressions. Back-slashes will always be interpreted as escape characters, not path separators.

Results from absolute patterns such as /foo/\* are mounted onto the root setting using path.join. On windows, this will by default result in /foo/\* matching C:\foo\bar.txt.

To automatically coerce all \ characters to / in pattern strings, **thus making it impossible to escape literal glob characters**, you may set the windowsPathsNoEscape option to true.

### Windows, CWDs, Drive Letters, and UNC Paths

On posix systems, when a pattern starts with /, any cwd option is ignored, and the traversal starts at /, plus any non-magic path portions specified in the pattern.

On Windows systems, the behavior is similar, but the concept of an "absolute path" is somewhat more involved.

#### UNC Paths

A UNC path may be used as the start of a pattern on Windows platforms. For example, a pattern like: //?/x:/\* will return all file entries in the root of the x: drive. A pattern like //ComputerName/Share/\* will return all files in the associated share.

UNC path roots are always compared case insensitively.

#### Drive Letters

A pattern starting with a drive letter, like c:/\*, will search in that drive, regardless of any cwd option provided.

If the pattern starts with /, and is not a UNC path, and there is an explicit cwd option set with a drive letter, then the drive letter in the cwd is used as the root of the directory traversal.

For example, glob('/tmp', { cwd: 'c:/any/thing' }) will return ['c:/tmp'] as the result.

If an explicit cwd option is not provided, and the pattern starts with /, then the traversal will run on the root of the drive provided as the cwd option. (That is, it is the result of path.resolve('/').)

## Race Conditions

Glob searching, by its very nature, is susceptible to race conditions, since it relies on directory walking.

As a result, it is possible that a file that exists when glob looks for it may have been deleted or modified by the time it returns the result.

By design, this implementation caches all readdir calls that it makes, in order to cut down on system overhead. However, this also makes it even more susceptible to races, especially if the cache object is reused between glob calls.

Users are thus advised not to use a glob result as a guarantee of filesystem state in the face of rapid changes. For the vast majority of operations, this is never a problem.

### See Also:

* man sh
* man bash [Pattern Matching](https://www.gnu.org/software/bash/manual/html_node/Pattern-Matching.html)
* man 3 fnmatch
* man 5 gitignore
* [minimatch documentation](https://github.com/isaacs/minimatch)

## Glob Logo

Glob's logo was created by [Tanya Brassie](http://tanyabrassie.com/). Logo files can be found [here](https://github.com/isaacs/node-glob/tree/master/logo).

The logo is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

## Contributing

Any change to behavior (including bugfixes) must come with a test.

Patches that fail tests or reduce performance will be rejected.

# to run tests

npm test

# to re-generate test fixtures

npm run test-regen

# run the benchmarks

npm run bench

# to profile javascript

npm run prof

## Comparison to Other JavaScript Glob Implementations

**tl;dr**

* If you want glob matching that is as faithful as possible to Bash pattern expansion semantics, and as fast as possible within that constraint, *use this module*.
* If you are reasonably sure that the patterns you will encounter are relatively simple, and want the absolutely fastest glob matcher out there, *use* [*fast-glob*](http://npm.im/fast-glob).
* If you are reasonably sure that the patterns you will encounter are relatively simple, and want the convenience of automatically respecting .gitignore files, *use* [*globby*](http://npm.im/globby).

There are some other glob matcher libraries on npm, but these three are (in my opinion, as of 2023) the best.

**full explanation**

Every library reflects a set of opinions and priorities in the trade-offs it makes. Other than this library, I can personally recommend both [globby](http://npm.im/globby) and [fast-glob](http://npm.im/fast-glob), though they differ in their benefits and drawbacks.

Both have very nice APIs and are reasonably fast.

fast-glob is, as far as I am aware, the fastest glob implementation in JavaScript today. However, there are many cases where the choices that fast-glob makes in pursuit of speed mean that its results differ from the results returned by Bash and other sh-like shells, which may be surprising.

In my testing, fast-glob is around 10-20% faster than this module when walking over 200k files nested 4 directories deep[1](#fn-webscale). However, there are some inconsistencies with Bash matching behavior that this module does not suffer from:

* \*\* only matches files, not directories
* .. path portions are not handled unless they appear at the start of the pattern
* ./!(<pattern>) will not match any files that *start* with <pattern>, even if they do not match <pattern>. For example, !(9).txt will not match 9999.txt.
* Some brace patterns in the middle of a pattern will result in failing to find certain matches.
* Extglob patterns are allowed to contain / characters.

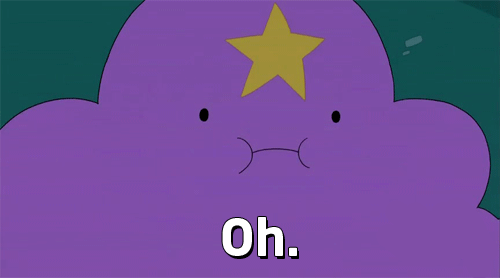
Globby exhibits all of the same pattern semantics as fast-glob, (as it is a wrapper around fast-glob) and is slightly slower than node-glob (by about 10-20% in the benchmark test set, or in other words, anywhere from 20-50% slower than fast-glob). However, it adds some API conveniences that may be worth the costs.

* Support for .gitignore and other ignore files.
* Support for negated globs (ie, patterns starting with ! rather than using a separate ignore option).

The priority of this module is "correctness" in the sense of performing a glob pattern expansion as faithfully as possible to the behavior of Bash and other sh-like shells, with as much speed as possible.

Note that prior versions of node-glob are *not* on this list. Former versions of this module are far too slow for any cases where performance matters at all, and were designed with APIs that are extremely dated by current JavaScript standards.

[1]: In the cases where this module returns results and `fast-glob` doesn't, it's even faster, of course.



### Benchmark Results

First number is time, smaller is better.

Second number is the count of results returned.

--- pattern: '\*\*' ---

~~ sync ~~

node fast-glob sync 0m0.598s 200364

node globby sync 0m0.765s 200364

node current globSync mjs 0m0.683s 222656

node current glob syncStream 0m0.649s 222656

~~ async ~~

node fast-glob async 0m0.350s 200364

node globby async 0m0.509s 200364

node current glob async mjs 0m0.463s 222656

node current glob stream 0m0.411s 222656

--- pattern: '\*\*/..' ---

~~ sync ~~

node fast-glob sync 0m0.486s 0

node globby sync 0m0.769s 200364

node current globSync mjs 0m0.564s 2242

node current glob syncStream 0m0.583s 2242

~~ async ~~

node fast-glob async 0m0.283s 0

node globby async 0m0.512s 200364

node current glob async mjs 0m0.299s 2242

node current glob stream 0m0.312s 2242

--- pattern: './\*\*/0/\*\*/0/\*\*/0/\*\*/0/\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.490s 10

node globby sync 0m0.517s 10

node current globSync mjs 0m0.540s 10

node current glob syncStream 0m0.550s 10

~~ async ~~

node fast-glob async 0m0.290s 10

node globby async 0m0.296s 10

node current glob async mjs 0m0.278s 10

node current glob stream 0m0.302s 10

--- pattern: './\*\*/[01]/\*\*/[12]/\*\*/[23]/\*\*/[45]/\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.500s 160

node globby sync 0m0.528s 160

node current globSync mjs 0m0.556s 160

node current glob syncStream 0m0.573s 160

~~ async ~~

node fast-glob async 0m0.283s 160

node globby async 0m0.301s 160

node current glob async mjs 0m0.306s 160

node current glob stream 0m0.322s 160

--- pattern: './\*\*/0/\*\*/0/\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.502s 5230

node globby sync 0m0.527s 5230

node current globSync mjs 0m0.544s 5230

node current glob syncStream 0m0.557s 5230

~~ async ~~

node fast-glob async 0m0.285s 5230

node globby async 0m0.305s 5230

node current glob async mjs 0m0.304s 5230

node current glob stream 0m0.310s 5230

--- pattern: '\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.580s 200023

node globby sync 0m0.771s 200023

node current globSync mjs 0m0.685s 200023

node current glob syncStream 0m0.649s 200023

~~ async ~~

node fast-glob async 0m0.349s 200023

node globby async 0m0.509s 200023

node current glob async mjs 0m0.427s 200023

node current glob stream 0m0.388s 200023

--- pattern: '{\*\*/\*.txt,\*\*/?/\*\*/\*.txt,\*\*/?/\*\*/?/\*\*/\*.txt,\*\*/?/\*\*/?/\*\*/?/\*\*/\*.txt,\*\*/?/\*\*/?/\*\*/?/\*\*/?/\*\*/\*.txt}' ---

~~ sync ~~

node fast-glob sync 0m0.589s 200023

node globby sync 0m0.771s 200023

node current globSync mjs 0m0.716s 200023

node current glob syncStream 0m0.684s 200023

~~ async ~~

node fast-glob async 0m0.351s 200023

node globby async 0m0.518s 200023

node current glob async mjs 0m0.462s 200023

node current glob stream 0m0.468s 200023

--- pattern: '\*\*/5555/0000/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.496s 1000

node globby sync 0m0.519s 1000

node current globSync mjs 0m0.539s 1000

node current glob syncStream 0m0.567s 1000

~~ async ~~

node fast-glob async 0m0.285s 1000

node globby async 0m0.299s 1000

node current glob async mjs 0m0.305s 1000

node current glob stream 0m0.301s 1000

--- pattern: './\*\*/0/\*\*/../[01]/\*\*/0/../\*\*/0/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.484s 0

node globby sync 0m0.507s 0

node current globSync mjs 0m0.577s 4880

node current glob syncStream 0m0.586s 4880

~~ async ~~

node fast-glob async 0m0.280s 0

node globby async 0m0.298s 0

node current glob async mjs 0m0.327s 4880

node current glob stream 0m0.324s 4880

--- pattern: '\*\*/????/????/????/????/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.547s 100000

node globby sync 0m0.673s 100000

node current globSync mjs 0m0.626s 100000

node current glob syncStream 0m0.618s 100000

~~ async ~~

node fast-glob async 0m0.315s 100000

node globby async 0m0.414s 100000

node current glob async mjs 0m0.366s 100000

node current glob stream 0m0.345s 100000

--- pattern: './{\*\*/?{/\*\*/?{/\*\*/?{/\*\*/?,,,,},,,,},,,,},,,}/\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.588s 100000

node globby sync 0m0.670s 100000

node current globSync mjs 0m0.717s 200023

node current glob syncStream 0m0.687s 200023

~~ async ~~

node fast-glob async 0m0.343s 100000

node globby async 0m0.418s 100000

node current glob async mjs 0m0.519s 200023

node current glob stream 0m0.451s 200023

--- pattern: '\*\*/!(0|9).txt' ---

~~ sync ~~

node fast-glob sync 0m0.573s 160023

node globby sync 0m0.731s 160023

node current globSync mjs 0m0.680s 180023

node current glob syncStream 0m0.659s 180023

~~ async ~~

node fast-glob async 0m0.345s 160023

node globby async 0m0.476s 160023

node current glob async mjs 0m0.427s 180023

node current glob stream 0m0.388s 180023

--- pattern: './{\*/\*\*/../{\*/\*\*/../{\*/\*\*/../{\*/\*\*/../{\*/\*\*,,,,},,,,},,,,},,,,},,,,}/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.483s 0

node globby sync 0m0.512s 0

node current globSync mjs 0m0.811s 200023

node current glob syncStream 0m0.773s 200023

~~ async ~~

node fast-glob async 0m0.280s 0

node globby async 0m0.299s 0

node current glob async mjs 0m0.617s 200023

node current glob stream 0m0.568s 200023

--- pattern: './\*/\*\*/../\*/\*\*/../\*/\*\*/../\*/\*\*/../\*/\*\*/../\*/\*\*/../\*/\*\*/../\*/\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.485s 0

node globby sync 0m0.507s 0

node current globSync mjs 0m0.759s 200023

node current glob syncStream 0m0.740s 200023

~~ async ~~

node fast-glob async 0m0.281s 0

node globby async 0m0.297s 0

node current glob async mjs 0m0.544s 200023

node current glob stream 0m0.464s 200023

--- pattern: './\*/\*\*/../\*/\*\*/../\*/\*\*/../\*/\*\*/../\*/\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.486s 0

node globby sync 0m0.513s 0

node current globSync mjs 0m0.734s 200023

node current glob syncStream 0m0.696s 200023

~~ async ~~

node fast-glob async 0m0.286s 0

node globby async 0m0.296s 0

node current glob async mjs 0m0.506s 200023

node current glob stream 0m0.483s 200023

--- pattern: './0/\*\*/../1/\*\*/../2/\*\*/../3/\*\*/../4/\*\*/../5/\*\*/../6/\*\*/../7/\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.060s 0

node globby sync 0m0.074s 0

node current globSync mjs 0m0.067s 0

node current glob syncStream 0m0.066s 0

~~ async ~~

node fast-glob async 0m0.060s 0

node globby async 0m0.075s 0

node current glob async mjs 0m0.066s 0

node current glob stream 0m0.067s 0

--- pattern: './\*\*/?/\*\*/?/\*\*/?/\*\*/?/\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.568s 100000

node globby sync 0m0.651s 100000

node current globSync mjs 0m0.619s 100000

node current glob syncStream 0m0.617s 100000

~~ async ~~

node fast-glob async 0m0.332s 100000

node globby async 0m0.409s 100000

node current glob async mjs 0m0.372s 100000

node current glob stream 0m0.351s 100000

--- pattern: '\*\*/\*/\*\*/\*/\*\*/\*/\*\*/\*/\*\*' ---

~~ sync ~~

node fast-glob sync 0m0.603s 200113

node globby sync 0m0.798s 200113

node current globSync mjs 0m0.730s 222137

node current glob syncStream 0m0.693s 222137

~~ async ~~

node fast-glob async 0m0.356s 200113

node globby async 0m0.525s 200113

node current glob async mjs 0m0.508s 222137

node current glob stream 0m0.455s 222137

--- pattern: './\*\*/\*/\*\*/\*/\*\*/\*/\*\*/\*/\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.622s 200000

node globby sync 0m0.792s 200000

node current globSync mjs 0m0.722s 200000

node current glob syncStream 0m0.695s 200000

~~ async ~~

node fast-glob async 0m0.369s 200000

node globby async 0m0.527s 200000

node current glob async mjs 0m0.502s 200000

node current glob stream 0m0.481s 200000

--- pattern: '\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.588s 200023

node globby sync 0m0.771s 200023

node current globSync mjs 0m0.684s 200023

node current glob syncStream 0m0.658s 200023

~~ async ~~

node fast-glob async 0m0.352s 200023

node globby async 0m0.516s 200023

node current glob async mjs 0m0.432s 200023

node current glob stream 0m0.384s 200023

--- pattern: './\*\*/\*\*/\*\*/\*\*/\*\*/\*\*/\*\*/\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.589s 200023

node globby sync 0m0.766s 200023

node current globSync mjs 0m0.682s 200023

node current glob syncStream 0m0.652s 200023

~~ async ~~

node fast-glob async 0m0.352s 200023

node globby async 0m0.523s 200023

node current glob async mjs 0m0.436s 200023

node current glob stream 0m0.380s 200023

--- pattern: '\*\*/\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.592s 200023

node globby sync 0m0.776s 200023

node current globSync mjs 0m0.691s 200023

node current glob syncStream 0m0.659s 200023

~~ async ~~

node fast-glob async 0m0.357s 200023

node globby async 0m0.513s 200023

node current glob async mjs 0m0.471s 200023

node current glob stream 0m0.424s 200023

--- pattern: '\*\*/\*/\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.585s 200023

node globby sync 0m0.766s 200023

node current globSync mjs 0m0.694s 200023

node current glob syncStream 0m0.664s 200023

~~ async ~~

node fast-glob async 0m0.350s 200023

node globby async 0m0.514s 200023

node current glob async mjs 0m0.472s 200023

node current glob stream 0m0.424s 200023

--- pattern: '\*\*/[0-9]/\*\*/\*.txt' ---

~~ sync ~~

node fast-glob sync 0m0.544s 100000

node globby sync 0m0.636s 100000

node current globSync mjs 0m0.626s 100000

node current glob syncStream 0m0.621s 100000

~~ async ~~

node fast-glob async 0m0.322s 100000

node globby async 0m0.404s 100000

node current glob async mjs 0m0.360s 100000

node current glob stream 0m0.352s 100000