# fast-glob

It's a very fast and efficient [glob](https://en.wikipedia.org/wiki/Glob_(programming)) library for [Node.js](https://nodejs.org/en).

This package provides methods for traversing the file system and returning pathnames that matched a defined set of a specified pattern according to the rules used by the Unix Bash shell with some simplifications, meanwhile results are returned in **arbitrary order**. Quick, simple, effective.

## Table of Contents

* **Details** [Highlights](#highlights)
* [Old and modern mode](#old-and-modern-mode)
* [Pattern syntax](#pattern-syntax)
  + [Basic syntax](#basic-syntax)
  + [Advanced syntax](#advanced-syntax)
* [Installation](#installation)
* [API](#api)
  + [Asynchronous](#asynchronous)
  + [Synchronous](#synchronous)
  + [Stream](#stream)
    - [patterns](#patterns)
    - [[options]](#options)
  + [Helpers](#helpers)
    - [generateTasks](#generatetaskspatterns-options)
    - [isDynamicPattern](#isdynamicpatternpattern-options)
    - [escapePath](#escapepathpath)
    - [convertPathToPattern](#convertpathtopatternpath)
* [Options](#options-3)
  + [Common](#common)
    - [concurrency](#concurrency)
    - [cwd](#cwd)
    - [deep](#deep)
    - [followSymbolicLinks](#followsymboliclinks)
    - [fs](#fs)
    - [ignore](#ignore)
    - [suppressErrors](#suppresserrors)
    - [throwErrorOnBrokenSymbolicLink](#throwerroronbrokensymboliclink)
  + [Output control](#output-control)
    - [absolute](#absolute)
    - [markDirectories](#markdirectories)
    - [objectMode](#objectmode)
    - [onlyDirectories](#onlydirectories)
    - [onlyFiles](#onlyfiles)
    - [stats](#stats)
    - [unique](#unique)
  + [Matching control](#matching-control)
    - [braceExpansion](#braceexpansion)
    - [caseSensitiveMatch](#casesensitivematch)
    - [dot](#dot)
    - [extglob](#extglob)
    - [globstar](#globstar)
    - [baseNameMatch](#basenamematch)
* [FAQ](#faq)
  + [What is a static or dynamic pattern?](#what-is-a-static-or-dynamic-pattern)
  + [How to write patterns on Windows?](#how-to-write-patterns-on-windows)
  + [Why are parentheses match wrong?](#why-are-parentheses-match-wrong)
  + [How to exclude directory from reading?](#how-to-exclude-directory-from-reading)
  + [How to use UNC path?](#how-to-use-unc-path)
  + [Compatible with node-glob?](#compatible-with-node-glob)
* [Benchmarks](#benchmarks)
  + [Server](#server)
  + [Nettop](#nettop)
* [Changelog](#changelog)
* [License](#license)

## Highlights

* Fast. Probably the fastest.
* Supports multiple and negative patterns.
* Synchronous, Promise and Stream API.
* Object mode. Can return more than just strings.
* Error-tolerant.

## Old and modern mode

This package works in two modes, depending on the environment in which it is used.

* **Old mode**. Node.js below 10.10 or when the [stats](#stats) option is *enabled*.
* **Modern mode**. Node.js 10.10+ and the [stats](#stats) option is *disabled*.

The modern mode is faster. Learn more about the [internal mechanism](https://github.com/nodelib/nodelib/blob/master/packages/fs/fs.scandir/README.md#old-and-modern-mode).

## Pattern syntax

:warning: Always use forward-slashes in glob expressions (patterns and [ignore](#ignore) option). Use backslashes for escaping characters.

There is more than one form of syntax: basic and advanced. Below is a brief overview of the supported features. Also pay attention to our [FAQ](#faq).

:book: This package uses [micromatch](https://github.com/micromatch/micromatch) as a library for pattern matching.

### Basic syntax

* An asterisk (\*) — matches everything except slashes (path separators), hidden files (names starting with .).
* A double star or globstar (\*\*) — matches zero or more directories.
* Question mark (?) – matches any single character except slashes (path separators).
* Sequence ([seq]) — matches any character in sequence.

:book: A few additional words about the [basic matching behavior](https://github.com/micromatch/picomatch#matching-behavior-vs-bash).

Some examples:

* src/\*\*/\*.js — matches all files in the src directory (any level of nesting) that have the .js extension.
* src/\*.?? — matches all files in the src directory (only first level of nesting) that have a two-character extension.
* file-[01].js — matches files: file-0.js, file-1.js.

### Advanced syntax

* [Escapes characters](https://github.com/micromatch/micromatch#backslashes) (\\) — matching special characters ($^\*+?()[]) as literals.
* [POSIX character classes](https://github.com/micromatch/picomatch#posix-brackets) ([[:digit:]]).
* [Extended globs](https://github.com/micromatch/micromatch#extglobs) (?(pattern-list)).
* [Bash style brace expansions](https://github.com/micromatch/braces) ({}).
* [Regexp character classes](https://github.com/micromatch/micromatch#regex-character-classes) ([1-5]).
* [Regex groups](https://www.regular-expressions.info/brackets.html) ((a|b)).

:book: A few additional words about the [advanced matching behavior](https://github.com/micromatch/micromatch#extended-globbing).

Some examples:

* src/\*\*/\*.{css,scss} — matches all files in the src directory (any level of nesting) that have the .css or .scss extension.
* file-[[:digit:]].js — matches files: file-0.js, file-1.js, …, file-9.js.
* file-{1..3}.js — matches files: file-1.js, file-2.js, file-3.js.
* file-(1|2) — matches files: file-1.js, file-2.js.

## Installation

npm install fast-glob

## API

### Asynchronous

fg(patterns, [options])

fg.async(patterns, [options])

fg.glob(patterns, [options])

Returns a Promise with an array of matching entries.

const fg = require('fast-glob');

const entries = await fg(['.editorconfig', '\*\*/index.js'], { dot: true });

// ['.editorconfig', 'services/index.js']

### Synchronous

fg.sync(patterns, [options])

fg.globSync(patterns, [options])

Returns an array of matching entries.

const fg = require('fast-glob');

const entries = fg.sync(['.editorconfig', '\*\*/index.js'], { dot: true });

// ['.editorconfig', 'services/index.js']

### Stream

fg.stream(patterns, [options])

fg.globStream(patterns, [options])

Returns a [ReadableStream](https://nodejs.org/api/stream.html#stream_readable_streams) when the data event will be emitted with matching entry.

const fg = require('fast-glob');

const stream = fg.stream(['.editorconfig', '\*\*/index.js'], { dot: true });

for await (const entry of stream) {

// .editorconfig

// services/index.js

}

#### patterns

* Required: true
* Type: string | string[]

Any correct pattern(s).

:1234: [Pattern syntax](#pattern-syntax)

:warning: This package does not respect the order of patterns. First, all the negative patterns are applied, and only then the positive patterns. If you want to get a certain order of records, use sorting or split calls.

#### [options]

* Required: false
* Type: [Options](#options-3)

See [Options](#options-3) section.

### Helpers

#### generateTasks(patterns, [options])

Returns the internal representation of patterns ([Task](http://./src/managers/tasks.ts) is a combining patterns by base directory).

fg.generateTasks('\*');

[{

base: '.', // Parent directory for all patterns inside this task

dynamic: true, // Dynamic or static patterns are in this task

patterns: ['\*'],

positive: ['\*'],

negative: []

}]

##### patterns

* Required: true
* Type: string | string[]

Any correct pattern(s).

##### [options]

* Required: false
* Type: [Options](#options-3)

See [Options](#options-3) section.

#### isDynamicPattern(pattern, [options])

Returns true if the passed pattern is a dynamic pattern.

:1234: [What is a static or dynamic pattern?](#what-is-a-static-or-dynamic-pattern)

fg.isDynamicPattern('\*'); // true

fg.isDynamicPattern('abc'); // false

##### pattern

* Required: true
* Type: string

Any correct pattern.

##### [options]

* Required: false
* Type: [Options](#options-3)

See [Options](#options-3) section.

#### escapePath(path)

Returns the path with escaped special characters depending on the platform.

* Posix:
  + \*?|(){}[];
  + ! at the beginning of line;
  + @+! before the opening parenthesis;
  + \\ before non-special characters;
* Windows:
  + (){}[]
  + ! at the beginning of line;
  + @+! before the opening parenthesis;
  + Characters like \*?| cannot be used in the path ([windows\_naming\_conventions](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file#naming-conventions)), so they will not be escaped;

fg.escapePath('!abc');

// \\!abc

fg.escapePath('[OpenSource] mrmlnc – fast-glob (Deluxe Edition) 2014') + '/\*.flac'

// \\[OpenSource\\] mrmlnc – fast-glob \\(Deluxe Edition\\) 2014/\*.flac

fg.posix.escapePath('C:\\Program Files (x86)\\\*\*\\\*');

// C:\\\\Program Files \\(x86\\)\\\*\\\*\\\*

fg.win32.escapePath('C:\\Program Files (x86)\\\*\*\\\*');

// Windows: C:\\Program Files \\(x86\\)\\\*\*\\\*

#### convertPathToPattern(path)

Converts a path to a pattern depending on the platform, including special character escaping.

* Posix. Works similarly to the fg.posix.escapePath method.
* Windows. Works similarly to the fg.win32.escapePath method, additionally converting backslashes to forward slashes in cases where they are not escape characters (!()+@{}[]).

fg.convertPathToPattern('[OpenSource] mrmlnc – fast-glob (Deluxe Edition) 2014') + '/\*.flac';

// \\[OpenSource\\] mrmlnc – fast-glob \\(Deluxe Edition\\) 2014/\*.flac

fg.convertPathToPattern('C:/Program Files (x86)/\*\*/\*');

// Posix: C:/Program Files \\(x86\\)/\\\*\\\*/\\\*

// Windows: C:/Program Files \\(x86\\)/\*\*/\*

fg.convertPathToPattern('C:\\Program Files (x86)\\\*\*\\\*');

// Posix: C:\\\\Program Files \\(x86\\)\\\*\\\*\\\*

// Windows: C:/Program Files \\(x86\\)/\*\*/\*

fg.posix.convertPathToPattern('\\\\?\\c:\\Program Files (x86)') + '/\*\*/\*';

// Posix: \\\\\\?\\\\c:\\\\Program Files \\(x86\\)/\*\*/\* (broken pattern)

fg.win32.convertPathToPattern('\\\\?\\c:\\Program Files (x86)') + '/\*\*/\*';

// Windows: //?/c:/Program Files \\(x86\\)/\*\*/\*

## Options

### Common options

#### concurrency

* Type: number
* Default: os.cpus().length

Specifies the maximum number of concurrent requests from a reader to read directories.

:book: The higher the number, the higher the performance and load on the file system. If you want to read in quiet mode, set the value to a comfortable number or 1.

More details In Node, there are [two types of threads](https://nodejs.org/en/docs/guides/dont-block-the-event-loop): Event Loop (code) and a Thread Pool (fs, dns, …). The thread pool size controlled by the UV\_THREADPOOL\_SIZE environment variable. Its default size is 4 ([documentation](http://docs.libuv.org/en/v1.x/threadpool.html)). The pool is one for all tasks within a single Node process.

Any code can make 4 real concurrent accesses to the file system. The rest of the FS requests will wait in the queue.

:book: Each new instance of FG in the same Node process will use the same Thread pool.

But this package also has the concurrency option. This option allows you to control the number of concurrent accesses to the FS at the package level. By default, this package has a value equal to the number of cores available for the current Node process. This allows you to set a value smaller than the pool size (concurrency: 1) or, conversely, to prepare tasks for the pool queue more quickly (concurrency: Number.POSITIVE\_INFINITY).

So, in fact, this package can **only make 4 concurrent requests to the FS**. You can increase this value by using an environment variable (UV\_THREADPOOL\_SIZE), but in practice this does not give a multiple advantage.

#### cwd

* Type: string
* Default: process.cwd()

The current working directory in which to search.

#### deep

* Type: number
* Default: Infinity

Specifies the maximum depth of a read directory relative to the start directory.

For example, you have the following tree:

dir/

└── one/ // 1

└── two/ // 2

└── file.js // 3

// With base directory

fg.sync('dir/\*\*', { onlyFiles: false, deep: 1 }); // ['dir/one']

fg.sync('dir/\*\*', { onlyFiles: false, deep: 2 }); // ['dir/one', 'dir/one/two']

// With cwd option

fg.sync('\*\*', { onlyFiles: false, cwd: 'dir', deep: 1 }); // ['one']

fg.sync('\*\*', { onlyFiles: false, cwd: 'dir', deep: 2 }); // ['one', 'one/two']

:book: If you specify a pattern with some base directory, this directory will not participate in the calculation of the depth of the found directories. Think of it as a [cwd](#cwd) option.

#### followSymbolicLinks

* Type: boolean
* Default: true

Indicates whether to traverse descendants of symbolic link directories when expanding \*\* patterns.

:book: Note that this option does not affect the base directory of the pattern. For example, if ./a is a symlink to directory ./b and you specified ['./a\*\*', './b/\*\*'] patterns, then directory ./a will still be read.

:book: If the [stats](#stats) option is specified, the information about the symbolic link (fs.lstat) will be replaced with information about the entry (fs.stat) behind it.

#### fs

* Type: FileSystemAdapter
* Default: fs.\*

Custom implementation of methods for working with the file system.

export interface FileSystemAdapter {

lstat?: typeof fs.lstat;

stat?: typeof fs.stat;

lstatSync?: typeof fs.lstatSync;

statSync?: typeof fs.statSync;

readdir?: typeof fs.readdir;

readdirSync?: typeof fs.readdirSync;

}

#### ignore

* Type: string[]
* Default: []

An array of glob patterns to exclude matches. This is an alternative way to use negative patterns.

dir/

├── package-lock.json

└── package.json

fg.sync(['\*.json', '!package-lock.json']); // ['package.json']

fg.sync('\*.json', { ignore: ['package-lock.json'] }); // ['package.json']

#### suppressErrors

* Type: boolean
* Default: false

By default this package suppress only ENOENT errors. Set to true to suppress any error.

:book: Can be useful when the directory has entries with a special level of access.

#### throwErrorOnBrokenSymbolicLink

* Type: boolean
* Default: false

Throw an error when symbolic link is broken if true or safely return lstat call if false.

:book: This option has no effect on errors when reading the symbolic link directory.

### Output control

#### absolute

* Type: boolean
* Default: false

Return the absolute path for entries.

fg.sync('\*.js', { absolute: false }); // ['index.js']

fg.sync('\*.js', { absolute: true }); // ['/home/user/index.js']

:book: This option is required if you want to use negative patterns with absolute path, for example, !${\_\_dirname}/\*.js.

#### markDirectories

* Type: boolean
* Default: false

Mark the directory path with the final slash.

fg.sync('\*', { onlyFiles: false, markDirectories: false }); // ['index.js', 'controllers']

fg.sync('\*', { onlyFiles: false, markDirectories: true }); // ['index.js', 'controllers/']

#### objectMode

* Type: boolean
* Default: false

Returns objects (instead of strings) describing entries.

fg.sync('\*', { objectMode: false }); // ['src/index.js']

fg.sync('\*', { objectMode: true }); // [{ name: 'index.js', path: 'src/index.js', dirent: <fs.Dirent> }]

The object has the following fields:

* name (string) — the last part of the path (basename)
* path (string) — full path relative to the pattern base directory
* dirent ([fs.Dirent](https://nodejs.org/api/fs.html#fs_class_fs_dirent)) — instance of fs.Dirent

:book: An object is an internal representation of entry, so getting it does not affect performance.

#### onlyDirectories

* Type: boolean
* Default: false

Return only directories.

fg.sync('\*', { onlyDirectories: false }); // ['index.js', 'src']

fg.sync('\*', { onlyDirectories: true }); // ['src']

:book: If true, the [onlyFiles](#onlyfiles) option is automatically false.

#### onlyFiles

* Type: boolean
* Default: true

Return only files.

fg.sync('\*', { onlyFiles: false }); // ['index.js', 'src']

fg.sync('\*', { onlyFiles: true }); // ['index.js']

#### stats

* Type: boolean
* Default: false

Enables an [object mode](#objectmode) with an additional field:

* stats ([fs.Stats](https://nodejs.org/api/fs.html#fs_class_fs_stats)) — instance of fs.Stats

fg.sync('\*', { stats: false }); // ['src/index.js']

fg.sync('\*', { stats: true }); // [{ name: 'index.js', path: 'src/index.js', dirent: <fs.Dirent>, stats: <fs.Stats> }]

:book: Returns fs.stat instead of fs.lstat for symbolic links when the [followSymbolicLinks](#followsymboliclinks) option is specified.

:warning: Unlike [object mode](#objectmode) this mode requires additional calls to the file system. On average, this mode is slower at least twice. See [old and modern mode](#old-and-modern-mode) for more details.

#### unique

* Type: boolean
* Default: true

Ensures that the returned entries are unique.

fg.sync(['\*.json', 'package.json'], { unique: false }); // ['package.json', 'package.json']

fg.sync(['\*.json', 'package.json'], { unique: true }); // ['package.json']

If true and similar entries are found, the result is the first found.

### Matching control

#### braceExpansion

* Type: boolean
* Default: true

Enables Bash-like brace expansion.

:1234: [Syntax description](https://wiki.bash-hackers.org/syntax/expansion/brace) or more [detailed description](https://github.com/micromatch/braces).

dir/

├── abd

├── acd

└── a{b,c}d

fg.sync('a{b,c}d', { braceExpansion: false }); // ['a{b,c}d']

fg.sync('a{b,c}d', { braceExpansion: true }); // ['abd', 'acd']

#### caseSensitiveMatch

* Type: boolean
* Default: true

Enables a [case-sensitive](https://en.wikipedia.org/wiki/Case_sensitivity) mode for matching files.

dir/

├── file.txt

└── File.txt

fg.sync('file.txt', { caseSensitiveMatch: false }); // ['file.txt', 'File.txt']

fg.sync('file.txt', { caseSensitiveMatch: true }); // ['file.txt']

#### dot

* Type: boolean
* Default: false

Allow patterns to match entries that begin with a period (.).

:book: Note that an explicit dot in a portion of the pattern will always match dot files.

dir/

├── .editorconfig

└── package.json

fg.sync('\*', { dot: false }); // ['package.json']

fg.sync('\*', { dot: true }); // ['.editorconfig', 'package.json']

#### extglob

* Type: boolean
* Default: true

Enables Bash-like extglob functionality.

:1234: [Syntax description](https://github.com/micromatch/micromatch#extglobs).

dir/

├── README.md

└── package.json

fg.sync('\*.+(json|md)', { extglob: false }); // []

fg.sync('\*.+(json|md)', { extglob: true }); // ['README.md', 'package.json']

#### globstar

* Type: boolean
* Default: true

Enables recursively repeats a pattern containing \*\*. If false, \*\* behaves exactly like \*.

dir/

└── a

└── b

fg.sync('\*\*', { onlyFiles: false, globstar: false }); // ['a']

fg.sync('\*\*', { onlyFiles: false, globstar: true }); // ['a', 'a/b']

#### baseNameMatch

* Type: boolean
* Default: false

If set to true, then patterns without slashes will be matched against the basename of the path if it contains slashes.

dir/

└── one/

└── file.md

fg.sync('\*.md', { baseNameMatch: false }); // []

fg.sync('\*.md', { baseNameMatch: true }); // ['one/file.md']

## FAQ

## What is a static or dynamic pattern?

All patterns can be divided into two types:

* **static**. A pattern is considered static if it can be used to get an entry on the file system without using matching mechanisms. For example, the file.js pattern is a static pattern because we can just verify that it exists on the file system.
* **dynamic**. A pattern is considered dynamic if it cannot be used directly to find occurrences without using a matching mechanisms. For example, the \* pattern is a dynamic pattern because we cannot use this pattern directly.

A pattern is considered dynamic if it contains the following characters (… — any characters or their absence) or options:

* The [caseSensitiveMatch](#casesensitivematch) option is disabled
* \\ (the escape character)
* \*, ?, ! (at the beginning of line)
* […]
* (…|…)
* @(…), !(…), \*(…), ?(…), +(…) (respects the [extglob](#extglob) option)
* {…,…}, {…..…} (respects the [braceExpansion](#braceexpansion) option)

## How to write patterns on Windows?

Always use forward-slashes in glob expressions (patterns and [ignore](#ignore) option). Use backslashes for escaping characters. With the [cwd](#cwd) option use a convenient format.

**Bad**

[

'directory\\\*',

path.join(process.cwd(), '\*\*')

]

**Good**

[

'directory/\*',

fg.convertPathToPattern(process.cwd()) + '/\*\*'

]

:book: Use the [.convertPathToPattern](#convertpathtopatternpath) package to convert Windows-style path to a Unix-style path.

Read more about [matching with backslashes](https://github.com/micromatch/micromatch#backslashes).

## Why are parentheses match wrong?

dir/

└── (special-\*file).txt

fg.sync(['(special-\*file).txt']) // []

Refers to Bash. You need to escape special characters:

fg.sync(['\\(special-\*file\\).txt']) // ['(special-\*file).txt']

Read more about [matching special characters as literals](https://github.com/micromatch/picomatch#matching-special-characters-as-literals). Or use the [.escapePath](#escapepathpath).

## How to exclude directory from reading?

You can use a negative pattern like this: !\*\*/node\_modules or !\*\*/node\_modules/\*\*. Also you can use [ignore](#ignore) option. Just look at the example below.

first/

├── file.md

└── second/

└── file.txt

If you don't want to read the second directory, you must write the following pattern: !\*\*/second or !\*\*/second/\*\*.

fg.sync(['\*\*/\*.md', '!\*\*/second']); // ['first/file.md']

fg.sync(['\*\*/\*.md'], { ignore: ['\*\*/second/\*\*'] }); // ['first/file.md']

:warning: When you write !\*\*/second/\*\*/\* it means that the directory will be **read**, but all the entries will not be included in the results.

You have to understand that if you write the pattern to exclude directories, then the directory will not be read under any circumstances.

## How to use UNC path?

You cannot use [Uniform Naming Convention (UNC)](https://learn.microsoft.com/openspecs/windows_protocols/ms-dtyp/62e862f4-2a51-452e-8eeb-dc4ff5ee33cc) paths as patterns (due to syntax) directly, but you can use them as [cwd](#cwd) directory or use the fg.convertPathToPattern method.

// cwd

fg.sync('\*', { cwd: '\\\\?\\C:\\Python27' /\* or //?/C:/Python27 \*/ });

fg.sync('Python27/\*', { cwd: '\\\\?\\C:\\' /\* or //?/C:/ \*/ });

// .convertPathToPattern

fg.sync(fg.convertPathToPattern('\\\\?\\c:\\Python27') + '/\*');

## Compatible with node-glob?

| **node-glob** | **fast-glob** |
| --- | --- |
| cwd | [cwd](#cwd) |
| root | – |
| dot | [dot](#dot) |
| nomount | – |
| mark | [markDirectories](#markdirectories) |
| nosort | – |
| nounique | [unique](#unique) |
| nobrace | [braceExpansion](#braceexpansion) |
| noglobstar | [globstar](#globstar) |
| noext | [extglob](#extglob) |
| nocase | [caseSensitiveMatch](#casesensitivematch) |
| matchBase | [baseNameMatch](#basenamematch) |
| nodir | [onlyFiles](#onlyfiles) |
| ignore | [ignore](#ignore) |
| follow | [followSymbolicLinks](#followsymboliclinks) |
| realpath | – |
| absolute | [absolute](#absolute) |

## Benchmarks

You can see results [here](https://github.com/mrmlnc/fast-glob/actions/workflows/benchmark.yml?query=branch%3Amaster) for every commit into the main branch.

* **Product benchmark** – comparison with the main competitors.
* **Regress benchmark** – regression between the current version and the version from the npm registry.

## Changelog

See the [Releases section of our GitHub project](https://github.com/mrmlnc/fast-glob/releases) for changelog for each release version.

## License

This software is released under the terms of the MIT license.