Explaining Globs

A glob is a string of literal and/or wildcard characters used to match filepaths. Globbing is the act of locating files on a filesystem using one or more globs.

The src() method expects a single glob string or an array of globs to determine which files your pipeline will operate on. At least one match must be found for your glob(s) otherwise src() will error. When an array of globs is used, they are matched in array order - especially useful for negative globs.

Segments and separators

A segment is everything between separators. The separator in a glob is always the / character - regardless of the operating system - even in Windows where the path separator is \\. In a glob, \\ is reserved as the escape character.

Here, the * is escaped, so it is treated as a literal instead of a wildcard character.

```
'glob_with_uncommon_\\*_character.js'
```

Avoid using Node's path methods, like path.join, to create globs. On Windows, it produces an invalid glob because Node uses \\ as the separator. Also avoid the __dirname global, __filename global, or process.cwd() for the same reasons.

```
const invalidGlob = path.join(__dirname, 'src/*.js');
```

Special character: * (single-star)

Matches any amount - including none - of characters within a single segment. Useful for globbing files within one directory.

This glob will match files like index.js, but not files like scripts/index.js or scripts/nested/index.js

```
'*.js'
```

Special character: ** (double-star)

Matches any amount - including none - of characters across segments. Useful for globbing files in nested directories. Make sure to appropriately restrict your double-star globs, to avoid matching large directories unnecessarily.
Here, the glob is appropriately restricted to the scripts/ directory. It will match files like scripts/index.js, scripts/nested/index.js, and scripts/nested/twice/index.js.

```
'scripts/**/*.js'
```

In the previous example, if `scripts/` wasn't prefixed, all dependencies in `node_modules` or other directories would also be matched.

Special character: ! (negative)

Since globs are matched in array order, a negative glob must follow at least one non-negative glob in an array. The first finds a set of matches, then the negative glob removes a portion of those results. When excluding all files within a directory, you must add /** after the directory name, which the globbing library optimizes internally.

```
['scripts/**/*.js', '!scripts/vendor/**']
```

If any non-negative globs follow a negative, nothing will be removed from the later set of matches.

```
['scripts/**/*.js', '!scripts/vendor/**', 'scripts/vendor/react.js']
```

Negative globs can be used as an alternative for restricting double-star globs.

```
['**/*.js', '!node_modules/**']
```

In the previous example, if the negative glob was `!node_modules/**/*.js`, the globbing library wouldn't optimize the negation and every match would have to be compared against the negative glob, which would be extremely

slow. To ignore all files in a directory, only add the `/**` glob after the directory name.

Overlapping globs

Two or more globs that (un)intentionally match the same file are considered overlapping. When overlapping globs are used within a single src(), gulp does its best to remove the duplicates, but doesn't attempt to deduplicate across separate src() calls.

Advanced resources

Most of what you'll need to work with globs in gulp is covered here. If you'd like to get more in depth, here are a few resources.

Micromatch Documentation
node-glob's Glob Primer
Begin's Globbing Documentation
Wikipedia's Glob Page