

[Discover Packages](#) > [Standard library](#) > [path](#) > [filepath](#) 

filepath





package

standard library

Version: [go1.20.1](#) **Latest** | Published: Feb 14, 2023 | License: [BSD-3-Clause](#) | Imports: 8 |

Imported by: 350,147

Details



- ✓ Valid [go.mod](#) file 
- ✓ Redistributable license 
- ✓ Tagged version 
- ✓ Stable version 

[Learn more](#)

Repository

[cs.opensource.google/go/go](#)

Links

 [Report a Vulnerability](#) Documentation 

<> Documentation

Rendered for [linux/amd64](#) 

Overview

Package filepath implements utility routines for manipulating filename paths in a way compatible with the target operating system-defined file paths.

The filepath package uses either forward slashes or backslashes, depending on the operating system. To process paths such as URLs that always use forward slashes regardless of the operating system, see the path package.

Index

[Constants](#)[Variables](#)[func Abs\(path string\) \(string, error\)](#)[func Base\(path string\) string](#)[func Clean\(path string\) string](#)[func Dir\(path string\) string](#)[func EvalSymlinks\(path string\) \(string, error\)](#)[func Ext\(path string\) string](#)[func FromSlash\(path string\) string](#)[func Glob\(pattern string\) \(matches \[\]string, err error\)](#)[func HasPrefix\(p, prefix string\) bool](#) **DEPRECATED**[func IsAbs\(path string\) bool](#)[func IsLocal\(path string\) bool](#)[func Join\(elem ...string\) string](#)

func Match(pattern, name string) (matched bool, err error)
func Rel(basepath, targpath string) (string, error)
func Split(path string) (dir, file string)
func SplitList(path string) []string
func ToSlash(path string) string
func VolumeName(path string) string
func Walk(root string, fn WalkFunc) error
func WalkDir(root string, fn fs.WalkDirFunc) error
type WalkFunc

Examples

Base
Dir
Ext
IsAbs
Join
Match
Rel
Split
SplitList
Walk

Constants

[View Source](#)

```
const (  
    Separator      = os.PathSeparator  
    ListSeparator = os.PathListSeparator  
)
```

Variables

[View Source](#)

```
var ErrBadPattern = errors.New("syntax error in pattern")
```

ErrBadPattern indicates a pattern was malformed.

[View Source](#)

```
var SkipAll error = fs.SkipAll
```

SkipAll is used as a return value from WalkFuncs to indicate that all remaining files and directories are to be skipped. It is not returned as an error by any function.

[View Source](#)

```
var SkipDir error = fs.SkipDir
```

SkipDir is used as a return value from WalkFuncs to indicate that the directory named in the call is to be skipped. It is not returned as an error by any function.

Functions

func Abs

```
func Abs(path string) (string, error)
```

Abs returns an absolute representation of path. If the path is not absolute it will be joined with the current working directory to turn it into an absolute path. The absolute path name for a given file is not guaranteed to be unique. Abs calls Clean on the result.

func Base

```
func Base(path string) string
```

Base returns the last element of path. Trailing path separators are removed before extracting the last element. If the path is empty, Base returns ".". If the path consists entirely of separators, Base returns a single separator.

► Example

func Clean

```
func Clean(path string) string
```

Clean returns the shortest path name equivalent to path by purely lexical processing. It applies the following rules iteratively until no further processing can be done:

1. Replace multiple Separator elements with a single one.
2. Eliminate each . path name element (the current directory).
3. Eliminate each inner .. path name element (the parent directory) along with the non-.. element that precedes it.
4. Eliminate .. elements that begin a rooted path: that is, replace "/.." by "/" at the beginning of a path, assuming Separator is '/'.

The returned path ends in a slash only if it represents a root directory, such as "/" on Unix or `C:\` on Windows.

Finally, any occurrences of slash are replaced by Separator.

If the result of this process is an empty string, Clean returns the string ".".

See also Rob Pike, "Lexical File Names in Plan 9 or Getting Dot-Dot Right," <https://9p.io/sys/doc/lexnames.html>

func Dir

```
func Dir(path string) string
```

Dir returns all but the last element of path, typically the path's directory. After dropping the final element, Dir calls Clean on the path and trailing slashes are removed. If the path is empty, Dir returns ".". If the path consists entirely of separators, Dir returns a single separator. The returned path does not end in a separator unless it is the root directory.

► [Example](#)

func EvalSymlinks

```
func EvalSymlinks(path string) (string, error)
```

EvalSymlinks returns the path name after the evaluation of any symbolic links. If path is relative the result will be relative to the current directory, unless one of the components is an absolute symbolic link. EvalSymlinks calls Clean on the result.

func Ext

```
func Ext(path string) string
```

Ext returns the file name extension used by path. The extension is the suffix beginning at the final dot in the final element of path; it is empty if there is no dot.

► [Example](#)

func FromSlash

```
func FromSlash(path string) string
```

FromSlash returns the result of replacing each slash ('/') character in path with a separator character. Multiple slashes are replaced by multiple separators.

func Glob

```
func Glob(pattern string) (matches []string, err error)
```

Glob returns the names of all files matching pattern or nil if there is no matching file. The syntax of patterns is the same as in Match. The pattern may describe hierarchical names such as /usr/*/bin/ed (assuming the Separator is '/').

Glob ignores file system errors such as I/O errors reading directories. The only possible returned error is ErrBadPattern, when pattern is malformed.

func HasPrefix DEPRECATED [Show](#)

func IsAbs

```
func IsAbs(path string) bool
```

IsAbs reports whether the path is absolute.

► [Example](#)

func IsLocal

added in go1.20

```
func IsLocal(path string) bool
```

IsLocal reports whether path, using lexical analysis only, has all of these properties:

- is within the subtree rooted at the directory in which path is evaluated
- is not an absolute path
- is not empty
- on Windows, is not a reserved name such as "NUL"

If IsLocal(path) returns true, then Join(base, path) will always produce a path contained within base and Clean(path) will always produce an unrooted path with no "." path elements.

IsLocal is a purely lexical operation. In particular, it does not account for the effect of any symbolic links that may exist in the filesystem.

func Join

```
func Join(elem ...string) string
```

Join joins any number of path elements into a single path, separating them with an OS specific Separator. Empty elements are ignored. The result is Cleaned. However, if the argument list is empty or all its elements are empty, Join returns an empty string. On Windows, the result will only be a UNC path if the first non-empty element is a UNC path.

► [Example](#)

func Match

```
func Match(pattern, name string) (matched bool, err error)
```

Match reports whether name matches the shell file name pattern. The pattern syntax is:

```
pattern:
    { term }
term:
    '*'           matches any sequence of non-Separator characters
```

```
'?'      matches any single non-Separator character
'[' [ '^' ] { character-range } ']'
          character class (must be non-empty)
c        matches character c (c != '*', '?', '\\', '[')
'\\' c    matches character c
```

character-range:

```
c        matches character c (c != '\\', '-', ']')
'\\' c    matches character c
lo '-' hi matches character c for lo <= c <= hi
```

Match requires pattern to match all of name, not just a substring. The only possible returned error is `ErrBadPattern`, when pattern is malformed.

On Windows, escaping is disabled. Instead, `\\` is treated as path separator.

► [Example](#)

func Rel

```
func Rel(basepath, targpath string) (string, error)
```

Rel returns a relative path that is lexically equivalent to targpath when joined to basepath with an intervening separator. That is, `Join(basepath, Rel(basepath, targpath))` is equivalent to targpath itself. On success, the returned path will always be relative to basepath, even if basepath and targpath share no elements. An error is returned if targpath can't be made relative to basepath or if knowing the current working directory would be necessary to compute it. Rel calls Clean on the result.

► [Example](#)

func Split

```
func Split(path string) (dir, file string)
```

Split splits path immediately following the final Separator, separating it into a directory and file name component. If there is no Separator in path, Split returns an empty dir and file set to path. The returned values have the property that `path = dir+file`.

► [Example](#)

func SplitList

```
func SplitList(path string) []string
```

SplitList splits a list of paths joined by the OS-specific ListSeparator, usually found in PATH or GOPATH environment variables. Unlike `strings.Split`, SplitList returns an empty slice when passed an empty string.

► Example

func ToSlash

```
func ToSlash(path string) string
```

ToSlash returns the result of replacing each separator character in path with a slash (/) character. Multiple separators are replaced by multiple slashes.

func VolumeName

```
func VolumeName(path string) string
```

VolumeName returns leading volume name. Given "C:\foo\bar" it returns "C:" on Windows. Given "\\host\share\foo" it returns "\\host\share". On other platforms it returns "".

func Walk

```
func Walk(root string, fn WalkFunc) error
```

Walk walks the file tree rooted at root, calling fn for each file or directory in the tree, including root.

All errors that arise visiting files and directories are filtered by fn: see the WalkFunc documentation for details.

The files are walked in lexical order, which makes the output deterministic but requires Walk to read an entire directory into memory before proceeding to walk that directory.

Walk does not follow symbolic links.

Walk is less efficient than WalkDir, introduced in Go 1.16, which avoids calling os.Lstat on every visited file or directory.

► Example

func WalkDir

added in go1.16

```
func WalkDir(root string, fn fs.WalkDirFunc) error
```

WalkDir walks the file tree rooted at root, calling fn for each file or directory in the tree, including root.

All errors that arise visiting files and directories are filtered by fn: see the fs.WalkDirFunc documentation for details.

The files are walked in lexical order, which makes the output deterministic but requires WalkDir to read an entire directory into memory before proceeding to walk that directory.

WalkDir does not follow symbolic links.

WalkDir calls fn with paths that use the separator character appropriate for the operating system. This is unlike [io/fs.WalkDir](#), which always uses slash separated paths.

Types

type WalkFunc

```
type WalkFunc func(path string, info fs.FileInfo, err error) error
```

WalkFunc is the type of the function called by Walk to visit each file or directory.

The path argument contains the argument to Walk as a prefix. That is, if Walk is called with root argument "dir" and finds a file named "a" in that directory, the walk function will be called with argument "dir/a".

The directory and file are joined with Join, which may clean the directory name: if Walk is called with the root argument "x/./dir" and finds a file named "a" in that directory, the walk function will be called with argument "dir/a", not "x/./dir/a".

The info argument is the fs.FileInfo for the named path.

The error result returned by the function controls how Walk continues. If the function returns the special value SkipDir, Walk skips the current directory (path if info.IsDir() is true, otherwise path's parent directory). If the function returns the special value SkipAll, Walk skips all remaining files and directories. Otherwise, if the function returns a non-nil error, Walk stops entirely and returns that error.

The err argument reports an error related to path, signaling that Walk will not walk into that directory. The function can decide how to handle that error; as described earlier, returning the error will cause Walk to stop walking the entire tree.

Walk calls the function with a non-nil err argument in two cases.

First, if an os.Lstat on the root directory or any directory or file in the tree fails, Walk calls the function with path set to that directory or file's path, info set to nil, and err set to the error from os.Lstat.

Second, if a directory's Readdirnames method fails, Walk calls the function with path set to the directory's path, info, set to an fs.FileInfo describing the directory, and err set to the error from Readdirnames.



Source Files

[View all](#)

[match.go](#)
[path.go](#)

[path_unix.go](#)
[symlink.go](#)

[symlink_unix.go](#)

Why Go

Get Started

Packages

About

Use Cases

Playground

Standard Library

Download

Case Studies

Tour

About Go Packages

Blog

[Stack Overflow](#)

[Issue Tracker](#)

[Help](#)

[Release Notes](#)

[Brand Guidelines](#)

[Code of Conduct](#)

Connect

[Twitter](#)

[GitHub](#)

[Slack](#)

[r/golang](#)

[Meetup](#)

[Golang Weekly](#)

[Copyright](#)

[Terms of Service](#)

[Privacy Policy](#)

[Report an Issue](#)



Google