# Package filepath go1.15.2 Latest

Published: **Sep 9, 2020** | License: BSD-3-Clause | Standard library

## 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.

## Constants

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

## Variables

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

ErrBadPattern indicates a pattern was malformed.

```
var SkipDir = errors.New("skip this directory")
```

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.

## 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.

## 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.

## 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.

## 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

```
func HasPrefix(p, prefix string) bool
```

HasPrefix exists for historical compatibility and should not be used.

Deprecated: HasPrefix does not respect path boundaries and does not ignore case when required.

## func IsAbs

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

IsAbs reports whether the path is absolute.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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, walkFn WalkFunc) error
```

Walk walks the file tree rooted at root, calling walkFn for each file or directory in the tree, including root. All errors that arise visiting files and directories are filtered by walkFn. The files are walked in lexical order, which makes the output deterministic but means that for very large directories Walk can be inefficient. Walk does not follow symbolic links.

## type WalkFunc

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

WalkFunc is the type of the function called for each file or directory visited by Walk. The path argument contains the argument to Walk as a prefix; that is, if Walk is called with "dir", which is a directory containing the file "a", the walk function will be called with argument "dir/a". The info argument is the os.FileInfo for the named path.

If there was a problem walking to the file or directory named by path, the incoming error will describe the problem and the function can decide how to handle that error (and Walk will not descend into that directory). In the case of an error, the info argument will be nil. If an error is returned, processing stops. The sole exception is when the function returns the special value SkipDir. If the function returns SkipDir when invoked on a directory, Walk skips the directory's contents entirely. If the function returns SkipDir when invoked on a non-directory file, Walk skips the remaining files in the containing directory.