

[Discover Packages](#) > [Standard library](#) > [io](#) > [fs](#) 





fs

package

standard library

Version: [go1.20.1](#) **Latest** | Published: Feb 14, 2023 | License: [BSD-3-Clause](#) | Imports: 7 |

Imported by: 10,769

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****Overview**

Package fs defines basic interfaces to a file system. A file system can be provided by the host operating system but also by other packages.

Index[Variables](#)[func Glob\(fsys FS, pattern string\) \(matches \[\]string, err error\)](#)[func ReadFile\(fsys FS, name string\) \(\[\]byte, error\)](#)[func ValidPath\(name string\) bool](#)[func WalkDir\(fsys FS, root string, fn WalkDirFunc\) error](#)[type DirEntry](#)[func FileInfoToDirEntry\(info FileInfo\) DirEntry](#)[func ReadDir\(fsys FS, name string\) \(\[\]DirEntry, error\)](#)[type FS](#)[func Sub\(fsys FS, dir string\) \(FS, error\)](#)[type File](#)[type FileInfo](#)[func Stat\(fsys FS, name string\) \(FileInfo, error\)](#)[type FileMode](#)[func \(m FileMode\) IsDir\(\) bool](#)[func \(m FileMode\) IsRegular\(\) bool](#)[func \(m FileMode\) Perm\(\) FileMode](#)[func \(m FileMode\) String\(\) string](#)

```
func (m FileMode) Type() FileMode
type GlobFS
type PathError
    func (e *PathError) Error() string
    func (e *PathError) Timeout() bool
    func (e *PathError) Unwrap() error
type ReadDirFS
type ReadDirFile
type ReadFileFS
type StatFS
type SubFS
type WalkDirFunc
```

Examples

[WalkDir](#)

Constants

This section is empty.

Variables

[View Source](#)

```
var (
    ErrInvalid      = errInvalid()    // "invalid argument"
    ErrPermission   = errPermission() // "permission denied"
    ErrExist        = errExist()      // "file already exists"
    ErrNotExist     = errNotExist()   // "file does not exist"
    ErrClosed       = errClosed()     // "file already closed"
)
```

Generic file system errors. Errors returned by file systems can be tested against these errors using `errors.Is`.

[View Source](#)

```
var SkipAll = errors.New("skip everything and stop the walk")
```

`SkipAll` is used as a return value from `WalkDirFuncs` 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 = errors.New("skip this directory")
```

`SkipDir` is used as a return value from `WalkDirFuncs` 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 Glob

```
func Glob(fsys FS, 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 path.Match. The pattern may describe hierarchical names such as usr/*/bin/ed.

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

If fs implements GlobFS, Glob calls fs.Glob. Otherwise, Glob uses ReadDir to traverse the directory tree and look for matches for the pattern.

func ReadFile

```
func ReadFile(fsys FS, name string) ([]byte, error)
```

ReadFile reads the named file from the file system fs and returns its contents. A successful call returns a nil error, not io.EOF. (Because ReadFile reads the whole file, the expected EOF from the final Read is not treated as an error to be reported.)

If fs implements ReadFileFS, ReadFile calls fs.ReadFile. Otherwise ReadFile calls fs.Open and uses Read and Close on the returned file.

func ValidPath

```
func ValidPath(name string) bool
```

ValidPath reports whether the given path name is valid for use in a call to Open.

Path names passed to open are UTF-8-encoded, unrooted, slash-separated sequences of path elements, like “x/y/z”. Path names must not contain an element that is “.” or “..” or the empty string, except for the special case that the root directory is named “.”. Paths must not start or end with a slash: “/x” and “x/” are invalid.

Note that paths are slash-separated on all systems, even Windows. Paths containing other characters such as backslash and colon are accepted as valid, but those characters must never be interpreted by an FS implementation as path element separators.

func WalkDir

```
func WalkDir(fsys FS, root string, fn 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 found in directories, but if root itself is a symbolic link, its target will be walked.

► [Example](#)

Types

type [DirEntry](#)

```
type DirEntry interface {
    // Name returns the name of the file (or subdirectory) described by the entry.
    // This name is only the final element of the path (the base name), not the entire path.
    // For example, Name would return "hello.go" not "home/gopher/hello.go".
    Name() string

    // IsDir reports whether the entry describes a directory.
    IsDir() bool

    // Type returns the type bits for the entry.
    // The type bits are a subset of the usual FileMode bits, those returned by the File.Mode() method.
    Type() FileMode

    // Info returns the FileInfo for the file or subdirectory described by the entry.
    // The returned FileInfo may be from the time of the original directory read
    // or from the time of the call to Info. If the file has been removed or renamed
    // since the directory read, Info may return an error satisfying errors.Is(err, ErrNotExist).
    // If the entry denotes a symbolic link, Info reports the information about the link
    // not the link's target.
    Info() (FileInfo, error)
}
```

A DirEntry is an entry read from a directory (using the ReadDir function or a ReadDirFile's ReadDir method).

func [FileInfoToDirEntry](#)

added in go1.17

```
func FileInfoToDirEntry(info FileInfo) DirEntry
```

FileInfoToDirEntry returns a DirEntry that returns information from info. If info is nil, FileInfoToDirEntry returns nil.

func [ReadDir](#)

```
func ReadDir(fsys FS, name string) ([]DirEntry, error)
```

ReadDir reads the named directory and returns a list of directory entries sorted by filename.

If `fs` implements `ReadDirFS`, `ReadDir` calls `fs.ReadDir`. Otherwise `ReadDir` calls `fs.Open` and uses `ReadDir` and `Close` on the returned file.

type FS

```
type FS interface {  
    // Open opens the named file.  
    //  
    // When Open returns an error, it should be of type *PathError  
    // with the Op field set to "open", the Path field set to name,  
    // and the Err field describing the problem.  
    //  
    // Open should reject attempts to open names that do not satisfy  
    // ValidPath(name), returning a *PathError with Err set to  
    // ErrInvalid or ErrNotExist.  
    Open(name string) (File, error)  
}
```

An FS provides access to a hierarchical file system.

The FS interface is the minimum implementation required of the file system. A file system may implement additional interfaces, such as `ReadFileFS`, to provide additional or optimized functionality.

func Sub

```
func Sub(fsys FS, dir string) (FS, error)
```

`Sub` returns an FS corresponding to the subtree rooted at `fsys`'s `dir`.

If `dir` is `"."`, `Sub` returns `fsys` unchanged. Otherwise, if `fs` implements `SubFS`, `Sub` returns `fsys.Sub(dir)`. Otherwise, `Sub` returns a new FS implementation `sub` that, in effect, implements `sub.Open(name)` as `fsys.Open(path.Join(dir, name))`. The implementation also translates calls to `ReadDir`, `ReadFile`, and `Glob` appropriately.

Note that `Sub(os.DirFS("/"), "prefix")` is equivalent to `os.DirFS("/prefix")` and that neither of them guarantees to avoid operating system accesses outside `"/prefix"`, because the implementation of `os.DirFS` does not check for symbolic links inside `"/prefix"` that point to other directories. That is, `os.DirFS` is not a general substitute for a chroot-style security mechanism, and `Sub` does not change that fact.

type File

```
type File interface {  
    Stat() (FileInfo, error)  
    Read([]byte) (int, error)  
    Close() error  
}
```

A File provides access to a single file. The File interface is the minimum implementation required of the file. Directory files should also implement `ReadDirFile`. A file may implement `io.ReaderAt` or `io.Seeker` as optimizations.

type FileInfo

```
type FileInfo interface {  
    Name() string          // base name of the file  
    Size() int64           // length in bytes for regular files; system-dependent for others  
    Mode() FileMode        // file mode bits  
    ModTime() time.Time    // modification time  
    IsDir() bool           // abbreviation for Mode().IsDir()  
    Sys() any              // underlying data source (can return nil)  
}
```

A FileInfo describes a file and is returned by Stat.

func Stat

```
func Stat(fsys FS, name string) (FileInfo, error)
```

Stat returns a FileInfo describing the named file from the file system.

If fs implements StatFS, Stat calls fs.Stat. Otherwise, Stat opens the file to stat it.

type FileMode

```
type FileMode uint32
```

A FileMode represents a file's mode and permission bits. The bits have the same definition on all systems, so that information about files can be moved from one system to another portably. Not all bits apply to all systems. The only required bit is ModeDir for directories.

```
const (  
    // The single letters are the abbreviations  
    // used by the String method's formatting.  
    ModeDir          FileMode = 1 << (32 - 1 - iota) // d: is a directory  
    ModeAppend                // a: append-only  
    ModeExclusive             // l: exclusive use  
    ModeTemporary            // T: temporary file; Plan 9 only  
    ModeSymlink              // L: symbolic link  
    ModeDevice               // D: device file  
    ModeNamedPipe            // p: named pipe (FIFO)  
    ModeSocket               // S: Unix domain socket  
    ModeSetuid               // u: setuid  
    ModeSetgid               // g: setgid  
    ModeCharDevice           // c: Unix character device, when ModeDevice is not set  
    ModeSticky               // t: sticky  
    ModeIrregular            // ?: non-regular file; nothing else  
  
    // Mask for the type bits. For regular files, none will be set.  
    ModeType = ModeDir | ModeSymlink | ModeNamedPipe | ModeSocket | ModeDevice | ModeCharDevice
```

```
ModePerm FileMode = 0777 // Unix permission bits
)
```

The defined file mode bits are the most significant bits of the FileMode. The nine least-significant bits are the standard Unix rwxrwxrwx permissions. The values of these bits should be considered part of the public API and may be used in wire protocols or disk representations: they must not be changed, although new bits might be added.

func (FileMode) IsDir

```
func (m FileMode) IsDir() bool
```

IsDir reports whether m describes a directory. That is, it tests for the ModeDir bit being set in m.

func (FileMode) IsRegular

```
func (m FileMode) IsRegular() bool
```

IsRegular reports whether m describes a regular file. That is, it tests that no mode type bits are set.

func (FileMode) Perm

```
func (m FileMode) Perm() FileMode
```

Perm returns the Unix permission bits in m (m & ModePerm).

func (FileMode) String

```
func (m FileMode) String() string
```

func (FileMode) Type

```
func (m FileMode) Type() FileMode
```

Type returns type bits in m (m & ModeType).

type GlobFS

```
type GlobFS interface {
    FS

    // Glob returns the names of all files matching pattern,
    // providing an implementation of the top-level
    // Glob function.
    Glob(pattern string) ([]string, error)
}
```

A GlobFS is a file system with a Glob method.

type **PathError**

```
type PathError struct {  
    Op    string  
    Path  string  
    Err   error  
}
```

PathError records an error and the operation and file path that caused it.

func (*PathError) **Error**

```
func (e *PathError) Error() string
```

func (*PathError) **Timeout**

```
func (e *PathError) Timeout() bool
```

Timeout reports whether this error represents a timeout.

func (*PathError) **Unwrap**

```
func (e *PathError) Unwrap() error
```

type **ReadDirFS**

```
type ReadDirFS interface {  
    FS  
  
    // ReadDir reads the named directory  
    // and returns a list of directory entries sorted by filename.  
    ReadDir(name string) ([]DirEntry, error)  
}
```

ReadDirFS is the interface implemented by a file system that provides an optimized implementation of ReadDir.

type **ReadDirFile**

```
type ReadDirFile interface {  
    File  
  
    // ReadDir reads the contents of the directory and returns  
    // a slice of up to n DirEntry values in directory order.  
    // Subsequent calls on the same file will yield further DirEntry values.  
    //  
    // If n > 0, ReadDir returns at most n DirEntry structures.  
    // In this case, if ReadDir returns an empty slice, it will return  
    // a non-nil error explaining why.
```



```
// At the end of a directory, the error is io.EOF.
// (ReadDir must return io.EOF itself, not an error wrapping io.EOF.)
//
// If n <= 0, ReadDir returns all the DirEntry values from the directory
// in a single slice. In this case, if ReadDir succeeds (reads all the way
// to the end of the directory), it returns the slice and a nil error.
// If it encounters an error before the end of the directory,
// ReadDir returns the DirEntry list read until that point and a non-nil error.
ReadDir(n int) ([]DirEntry, error)
}
```

A `ReadDirFile` is a directory file whose entries can be read with the `ReadDir` method. Every directory file should implement this interface. (It is permissible for any file to implement this interface, but if so `ReadDir` should return an error for non-directories.)

type `ReadFileFS`

```
type ReadFileFS interface {
    FS

    // ReadFile reads the named file and returns its contents.
    // A successful call returns a nil error, not io.EOF.
    // (Because ReadFile reads the whole file, the expected EOF
    // from the final Read is not treated as an error to be reported.)
    //
    // The caller is permitted to modify the returned byte slice.
    // This method should return a copy of the underlying data.
    ReadFile(name string) ([]byte, error)
}
```

`ReadFileFS` is the interface implemented by a file system that provides an optimized implementation of `ReadFile`.

type `StatFS`

```
type StatFS interface {
    FS

    // Stat returns a FileInfo describing the file.
    // If there is an error, it should be of type *PathError.
    Stat(name string) (FileInfo, error)
}
```

A `StatFS` is a file system with a `Stat` method.

type `SubFS`

```
type SubFS interface {
    FS

    // Sub returns an FS corresponding to the subtree rooted at dir.
```

```
Sub(dir string) (FS, error)
}
```

A SubFS is a file system with a Sub method.

type WalkDirFunc

```
type WalkDirFunc func(path string, d DirEntry, err error) error
```

WalkDirFunc is the type of the function called by WalkDir to visit each file or directory.

The path argument contains the argument to WalkDir as a prefix. That is, if WalkDir 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 d argument is the fs.DirEntry for the named path.

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

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

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

First, if the initial fs.Stat on the root directory fails, WalkDir calls the function with path set to root, d set to nil, and err set to the error from fs.Stat.

Second, if a directory's ReadDir method fails, WalkDir calls the function with path set to the directory's path, d set to an fs.DirEntry describing the directory, and err set to the error from ReadDir. In this second case, the function is called twice with the path of the directory: the first call is before the directory read is attempted and has err set to nil, giving the function a chance to return SkipDir or SkipAll and avoid the ReadDir entirely. The second call is after a failed ReadDir and reports the error from ReadDir. (If ReadDir succeeds, there is no second call.)

The differences between WalkDirFunc compared to filepath.WalkFunc are:

- The second argument has type fs.DirEntry instead of fs.FileInfo.
- The function is called before reading a directory, to allow SkipDir or SkipAll to bypass the directory read entirely or skip all remaining files and directories respectively.
- If a directory read fails, the function is called a second time for that directory to report the error.

Source Files

[View all](#) 

[fs.go](#)
[glob.go](#)

[readdir.go](#)
[readfile.go](#)

[stat.go](#)
[sub.go](#)

Why Go

[Use Cases](#)

[Case Studies](#)

Get Started

[Playground](#)

[Tour](#)

[Stack Overflow](#)

[Help](#)

Packages

[Standard Library](#)

[About Go Packages](#)

About

[Download](#)

[Blog](#)

[Issue Tracker](#)

[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