fsnotify / fsnotify Public

Cross-platform file system notifications for Go.

BSD-3-Clause license

7.8k stars **811** forks

Star Watch

Code Issues 24 Pull requests 8 Actions Security Insights

main

arp242 … ✓ 2 weeks ago

View code

fsnotify is a Go library to provide cross-platform filesystem notifications on Windows, Linux, macOS, BSD, and illumos.

Go 1.16 or newer is required; the full documentation is at https://pkg.go.dev/github.com/fsnotify/fsnotify

Platform support:

Backend	**OS**	**Status**
inotify	Linux	Supported
kqueue	BSD, macOS	Supported
ReadDirectoryChangesW	Windows	Supported
FEN	illumos	Supported in main branch
fanotify	Linux 5.9+	Not yet
AHAFS	AIX	aix branch; experimental due to lack of maintainer and test environment
FSEvents	macOS	Needs support in x/sys/unix

Backend	OS	Status
USN Journals	Windows	Needs support in x/sys/windows
Polling	*All*	Not yet

Linux and illumos should include Android and Solaris, but these are currently untested.

Usage

A basic example:

```
package main

import (
    "log"

    "github.com/fsnotify/fsnotify"
)

func main() {
    // Create new watcher.
    watcher, err := fsnotify.NewWatcher()
    if err != nil {
        log.Fatal(err)
    }
    defer watcher.Close()

    // Start listening for events.
    go func() {
        for {
            select {
            case event, ok := <-watcher.Events:
                if !ok {
                    return
                }
                log.Println("event:", event)
                if event.Has(fsnotify.Write) {
                    log.Println("modified file:", event.Name)
                }
            case err, ok := <-watcher.Errors:
                if !ok {
                    return
                }
                log.Println("error:", err)
            }
        }
    }()

    // Add a path.
    err = watcher.Add("/tmp")
    if err != nil {
        log.Fatal(err)
    }

    // Block main goroutine forever.
    <-make(chan struct{})
}
```

Some more examples can be found in cmd/fsnotify, which can be run with:

```
% go run ./cmd/fsnotify
```

Further detailed documentation can be found in godoc:

README.md

FAQ

Will a file still be watched when it's moved to another directory?

No, not unless you are watching the location it was moved to.

Are subdirectories watched too?

No, you must add watches for any directory you want to watch (a recursive watcher is on the roadmap: #18).

Do I have to watch the Error and Event channels in a goroutine?

As of now, yes (you can read both channels in the same goroutine using `select` , you don't need a separate goroutine for both channels; see the example).

Why don't notifications work with NFS, SMB, FUSE, /proc, or /sys?

fsnotify requires support from underlying OS to work. The current NFS and SMB protocols does not provide network level support for file notifications, and neither do the /proc and /sys virtual filesystems.

This could be fixed with a polling watcher (#9), but it's not yet implemented.

Why do I get many Chmod events?

Some programs may generate a lot of attribute changes; for example Spotlight on macOS, anti-virus programs, backup applications, and some others are known to do this. As a rule, it's typically best to ignore Chmod events. They're often not useful, and tend to cause problems.

Spotlight indexing on macOS can result in multiple events (see #15). A temporary workaround is to add your folder(s) to the *Spotlight Privacy settings* until we have a native FSEvents implementation (see #11).

Platform-specific notes

Linux

When a file is removed a REMOVE event won't be emitted until all file descriptors are closed; it will emit a CHMOD instead:

```
fp := os.Open("file")
os.Remove("file")        // CHMOD
fp.Close()               // REMOVE
```

This is the event that inotify sends, so not much can be changed about this.

The `fs.inotify.max_user_watches` sysctl variable specifies the upper limit for the number of watches per user, and `fs.inotify.max_user_instances` specifies the maximum number of inotify instances per user. Every Watcher you create is an "instance", and every path you add is a "watch".

These are also exposed in `/proc` as `/proc/sys/fs/inotify/max_user_watches` and `/proc/sys/fs/inotify/max_user_instances`

To increase them you can use `sysctl` or write the value to proc file:

```
# The default values on Linux 5.18
sysctl fs.inotify.max_user_watches=124983
sysctl fs.inotify.max_user_instances=128
```

To make the changes persist on reboot edit `/etc/sysctl.conf` or `/usr/lib/sysctl.d/50-default.conf` (details differ per Linux distro; check your distro's documentation):

```
fs.inotify.max_user_watches=124983
fs.inotify.max_user_instances=128
```

Reaching the limit will result in a "no space left on device" or "too many open files" error.

kqueue (macOS, all BSD systems)

kqueue requires opening a file descriptor for every file that's being watched; so if you're watching a directory with five files then that's six file descriptors. You will run in to your system's "max open files" limit faster on these platforms.

The sysctl variables `kern.maxfiles` and `kern.maxfilesperproc` can be used to control the maximum number of open files.

Releases 27

v1.6.0 Latest
on Oct 13, 2022

+ 26 releases

Sponsor this project

arp242 Martin Tournoij

Sponsor

Learn more about GitHub Sponsors

Used by 158k

+ 157,789

Contributors 68

+ 57 contributors

Languages

● **Go** 93.5% ● **Shell** 5.0% ● **C** 1.5%