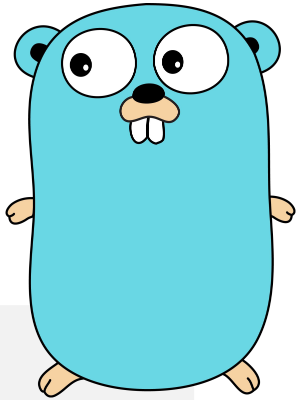
tail -f



Fabian Stäber
Munich Gophers Meetup
13 October 2016

Copy-and-Paste from Go Tutorial

```
func main() {
    file, err := os.Open("file.log")
    if err != nil {
        log.Fatal(err)
    }
    defer file.Close()
    reader := bufio.NewReader(file)
    for {
        line, err := reader.ReadString('\n')
        if err != nil {
            break
        }
        fmt.Print(string(line))
}   }
```

Copy-and-Paste from Go Tutorial

```
func main() {
    file, err := os.Open("file.log")
    if err != nil {
        log.Fatal(err)
    }
    defer file.Close()
    reader := bufio.NewReader(file)
    for {
        line, err := reader.ReadString('\n')
        if err != nil {
            break
        }
        fmt.Print(string(line))
}   }
```

This implements ‘cat’.

How to extend this to ‘tail -f’?

Copy-and-Paste from Go Tutorial

```
func main() {
    // ...
    for {
        line, err := reader.ReadString('\n')
        if err != nil {
            break
        }
        fmt.Print(string(line))
}   }
```

My first ‘tail -f’

```
func main() {
    // ...
    for {
        line, err := reader.ReadString('\n')
        if err != nil {
            if err == io.EOF {
                time.Sleep(1 * time.Second)
            } else {
                break
            }
        }
        fmt.Print(string(line))
}   }
```

Logrotate

Demo: Execute these commands while the tailer is running.

```
mv file.log file.log.1
echo example log line >> file.log
echo example log line >> file.log.1
rm file.log.1
```

Note: Cannot do this on Windows, because the file is “in use by another process”.

Logrotate Configurations

Move the old file and create a new one.

```
mv logfile logfile.1
:> logfile
echo 'next log line' >> logfile
```

Copy the old file and truncate the original copy.

```
cp logfile logfile.1
:> logfile
echo 'next log line' >> logfile
```

How to deal with logrotate

```
if err == io.EOF {
    time.Sleep(1 * time.Second)
    // check if file was truncated
    // check if file was moved
    // close and re-open if necessary
}
```

$\uparrow$

Need to do this.

How to deal with logrotate

```
if err == io.EOF {
    time.Sleep(1 * time.Second)
    // check if file was truncated
    // check if file was moved
    // close and re-open if necessary
}
```

Need to do this.

How to do it.

```
fileInfo, err = file.Stat()
if fileInfo.Size() < curReadPos {
    // truncated
}
if ! os.SameFile(fileInfo, curFileInfo) {
    // moved
}
```

FileBeat: github.com/elastic/beats

Pro

- Reasonable and simple

Con

- Requires weird trade-offs in configuration

  ```
  backoff, backoff_factor, max_backoff
  close_eof, close_inactive, close_older, close_removed, close_renamed
  ```

- There must be a way to do this without polling

fsnotify: github.com/fsnotify/fsnotify

Abstract File System notifications.

Used in:

- github.com/hpcloud/tail
- github.com/google/mtail

fsnotify: github.com/fsnotify/fsnotify

```
watcher, err := fsnotify.NewWatcher()
go func() {
    for {
        select {
        case event := <-watcher.Events:
            // ...
        case err := <-watcher.Errors:
            // ...
        }
    }
}
watcher.Add("file.log")
```

Events:

- Create
- Write
- Remove
- Chmod

fsnotify: github.com/fsnotify/fsnotify

Linux inotify() system call mapping:

unix.IN_CREATE, unix.IN_MOVED_TO	fsnotify.Create
unix.IN_DELETE_SELF, unix.IN_DELETE	fsnotify.Remove
unix.IN_MODIFY	fsnotify.Write
unix.IN_MOVE_SELF, unix.IN_MOVED_FROM	fsnotify.Rename
unix.IN_ATTRIB	fsnotify.Chmod
others	ignored

truncate

Fsnotify watches the **directory** where the log file is located.

fsnotify: github.com/fsnotify/fsnotify

BSD kevent() system call mapping:

???	fsnotify.Create
unix.NOTE_DELETE	fsnotify.Remove
unix.NOTE_WRITE	fsnotify.Write
unix.NOTE_RENAME	fsnotify.Rename
unix.NOTE_ATTRIB	fsnotify.Chmod $\leftarrow$ truncate
others	ignored

Fsnotify “simulates” recursive directory watches.

fsnotify: github.com/fsnotify/fsnotify

OS-specific code needed:

- Make up for missing WRITE events due to races on BSD
- Figure out if file was truncated on fsnotify.Write and fsnotify.Chmod
- Must close file on Windows, can only watch open files on BSD?
- ...

OS-specific corner cases must be found with trial-and-error.
Rigid testing needed, use Travis CI for OS X and Linux, AppVeyor for Windows.

Most fsnotify-based tools ignore this and focus on Linux.

fsnotify: github.com/fsnotify/fsnotify

OS specific file system events → OS independent fsnotify events → OS specific interpretation of fsnotify events

Better use system calls directly.

BSD and Linux example

BSD: kevent()

```
fd = syscall.Kqueue()


go func() {
    for {
        syscall.Kevent(fd, ...)
        // ...
        logData <- data
    }
}
```

Linux: inotify()

```
fd = syscall.InotifyInit1(...)
syscall.InotifyAddWatch(fd, ...)

go func() {
    for {
        syscall.Read(fd, ...)
        // ...
        logData <- data
    }
}
```

How to shut down

Producer: Event loop

```
go func() {
     for {
          err = syscall.Read(fd, ...)
          if err == interrupted {
               return
          }
          // ...
          logData <- data
     }
}
```

Consumer: Log line processor

```
select {
     case data := <- logData:
          // do something with data
     case <- quit:
          // interrupt the system call
}
```

How to interrupt read() ?

How to shut down

BSD: kevent()

```
syscall.Close(fd)
```

Linux: inotify()

```
syscall.InotifyRmWatch(fd, ...)
```

How to shut down

Producer: Event loop

```
go func() {
    for {
        err = syscall.Read(fd, ...)
        if err == interrupted {
            return
        }
        // ...
        logData <- data
    }
}
```

Consumer: Log line processor

```
select {
    case data := <- logData:
        // do something with data
    case <- quit:
        // interrupt the system call
}
```

How to shut down

Producer: Event loop

```
go func() {
    for {
        err = syscall.Read(fd, ...)
        if err == interrupted {
            return
        }
        // ...
        logData <- data
    }
}
```

Consumer: Log line processor

```
select {
    case data := <- logData:
        // do something with data
    case <- quit:
        // interrupt the system call
}
```

What if the producer hangs here when read() is interrupted?

How to shut down

Producer: Event loop

```
go func() {
    for {
        // ...
        logData <- data
    }
}
```

Consumer: Log line processor

```
select {
    case data := <- logData:
        // do something with data
    case <- quit:
        // interrupt the system call
}
```

How to shut down

Producer: Event loop

```
go func() {
    for {
        // ...
        select {
            case logData <- data:
            case <- done:
                return
        }
    }
}
```

Consumer: Log line processor

```
select {
    case data := <- logData:
        // do something with data
    case <- quit:
        close(done)
        // interrupt the system call
}
```

Thank you!

github.com/fstab/grok_exporter
package tailer/

@fstabr

http://www.consol.de