lumberjack

Lumberjack is a Go package for writing logs to rolling files.

Package lumberjack provides a rolling logger.

Note that this is v2.0 of lumberjack, and should be imported using gopkg.in thusly:

```
import "gopkg.in/natefinch/lumberjack.v2"
```

The package name remains simply lumberjack, and the code resides at https://github.com/natefinch/lumberjack under the v2.0 branch.

Lumberjack is intended to be one part of a logging infrastructure. It is not an all-in-one solution, but instead is a pluggable component at the bottom of the logging stack that simply controls the files to which logs are written.

Lumberjack plays well with any logging package that can write to an io.Writer, including the standard library's log package.

Lumberjack assumes that only one process is writing to the output files. Using the same lumberjack configuration from multiple processes on the same machine will result in improper behavior.

Example

To use lumberjack with the standard library's log package, just pass it into the SetOutput function when your application starts.

Code:

```
log.SetOutput(&lumberjack.Logger{
    Filename:   "/var/log/myapp/foo.log",
    MaxSize:    500, // megabytes
    MaxBackups: 3,
    MaxAge:     28, //days
    Compress:   true, // disabled by default
})
```

type Logger

```
type Logger struct {
    // Filename is the file to write logs to.  Backup log files will be retained
    // in the same directory.  It uses <processname>-lumberjack.log in
    // os.TempDir() if empty.
    Filename string `json:"filename" yaml:"filename"`

    // MaxSize is the maximum size in megabytes of the log file before it gets
    // rotated. It defaults to 100 megabytes.
    MaxSize int `json:"maxsize" yaml:"maxsize"`
```

```
    // MaxAge is the maximum number of days to retain old log files based on the
    // timestamp encoded in their filename.  Note that a day is defined as 24
    // hours and may not exactly correspond to calendar days due to daylight
    // savings, leap seconds, etc. The default is not to remove old log files
    // based on age.
    MaxAge int `json:"maxage" yaml:"maxage"`

    // MaxBackups is the maximum number of old log files to retain.  The default
    // is to retain all old log files (though MaxAge may still cause them to get
    // deleted.)
    MaxBackups int `json:"maxbackups" yaml:"maxbackups"`

    // LocalTime determines if the time used for formatting the timestamps in
    // backup files is the computer's local time.  The default is to use UTC
    // time.
    LocalTime bool `json:"localtime" yaml:"localtime"`

    // Compress determines if the rotated log files should be compressed
    // using gzip. The default is not to perform compression.
    Compress bool `json:"compress" yaml:"compress"`
    // contains filtered or unexported fields
}
```

Logger is an io.WriteCloser that writes to the specified filename.

Logger opens or creates the logfile on first Write. If the file exists and is less than MaxSize megabytes, lumberjack will open and append to that file. If the file exists and its size is >= MaxSize megabytes, the file is renamed by putting the current time in a timestamp in the name immediately before the file's extension (or the end of the filename if there's no extension). A new log file is then created using original filename.

Whenever a write would cause the current log file exceed MaxSize megabytes, the current file is closed, renamed, and a new log file created with the original name. Thus, the filename you give Logger is always the "current" log file.

Backups use the log file name given to Logger, in the form `name-timestamp.ext` where name is the filename without the extension, timestamp is the time at which the log was rotated formatted with the time.Time format of `2006-01-02T15-04-05.000` and the extension is the original extension. For example, if your Logger.Filename is `/var/log/foo/server.log`, a backup created at 6:30pm on Nov 11 2016 would use the filename `/var/log/foo/server-2016-11-04T18-30-00.000.log`

Cleaning Up Old Log Files

Whenever a new logfile gets created, old log files may be deleted. The most recent files according to the encoded timestamp will be retained, up to a number equal to MaxBackups (or all of them if MaxBackups is 0). Any files with an encoded timestamp older than MaxAge days are deleted, regardless of MaxBackups. Note that the time encoded in the timestamp is the rotation time, which may differ from the last time that file was written to.

If MaxBackups and MaxAge are both 0, no old log files will be deleted.

func (*Logger) Close

```
func (l *Logger) Close() error
```

Close implements io.Closer, and closes the current logfile.

func (*Logger) Rotate

```
func (l *Logger) Rotate() error
```

Rotate causes Logger to close the existing log file and immediately create a new one. This is a helper function for applications that want to initiate rotations outside of the normal rotation rules, such as in response to SIGHUP. After rotating, this initiates a cleanup of old log files according to the normal rules.

Example

Example of how to rotate in response to SIGHUP.

Code:

```
l := &lumberjack.Logger{}
log.SetOutput(l)
c := make(chan os.Signal, 1)
signal.Notify(c, syscall.SIGHUP)

go func() {
    for {
        <-c
        l.Rotate()
    }
}()
```

func (*Logger) Write

```
func (l *Logger) Write(p []byte) (n int, err error)
```

Write implements io.Writer. If a write would cause the log file to be larger than MaxSize, the file is closed, renamed to include a timestamp of the current time, and a new log file is created using the original log file name. If the length of the write is greater than MaxSize, an error is returned.
