

```

/**
 * dirnotify - a command-line utility for directory notifications.
 *
 * Fredrik Fornwall (fredrikfornwall@gmail.com)
 */

#include <ctime>
#include <iomanip>
#include <iostream>
#include <map>
#include <sstream>
#include <string>
#include <vector>

#include <dirent.h>
#include <errno.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>

using std::cout;
using std::endl;
using std::string;

struct stat scratchStat;

struct Settings {
    unsigned long eventMask;
    bool justDie;
    bool timestamp;
    bool silent;
    void initialize() {
        eventMask = DN_MULTISHOT;
        justDie = false;
        timestamp = false;
        silent = false;
    };
} settings;

std::ostream& timestamp(std::ostream& out) {
    if (settings.timestamp) {
        static char buffer[100];
        std::time_t time = std::time(NULL);
        std::tm* localTime = std::localtime(&time);
        std::strftime(buffer, sizeof(buffer), "%H:%M:%S", localTime);
        out << buffer << " ";
    }
    return out;
}

struct File {
    File() {}
    File(struct stat fileStat, std::string name, std::string fullPath) : fileStat(fileStat),
name(name), fullPath(fullPath) {}
    struct stat fileStat;
    std::string name;
    std::string fullPath;

    void update() {
        bool resized = false;
        std::stringstream ss;
        if ((scratchStat.st_atim.tv_sec != fileStat.st_atim.tv_sec) or (scratchStat.st_atim.tv_nsec
!= fileStat.st_atim.tv_nsec)) {
            ss << " read";
        }
        if (scratchStat.st_mode != fileStat.st_mode) {
            ss << " mode[" << std::setbase(8) << std::setw(6) << std::setfill('0')
<< fileStat.st_mode << "=>" << scratchStat.st_mode << " ]";
        }
    }
};

```

```

    }
    if (scratchStat.st_size != fileStat.st_size) {
        resized = true;
        ss << " resized[" << fileStat.st_size << "=>" << scratchStat.st_size << "];"
    }
    if (scratchStat.st_uid != fileStat.st_uid) {
        ss << " owner[" << fileStat.st_uid << "=>" << scratchStat.st_uid << "];"
    }
    if (scratchStat.st_mtime != fileStat.st_mtime) {
        if (not resized) {
            ss << " changedcontent";
        }
    }
}

if (ss.str().size() != 0U) {
    if (settings.silent) {
        cout << name << endl;
    } else {
        timestamp(cout) << name << ss.str() << endl;
    }
}
fileStat = scratchStat;
}
};

std::map<ino_t, File> fileMap;

static void signalHandler(int sig, siginfo_t* si, void* data) { }

void dieExplaining() {
    std::cout << "Usage: dirnotify [-e event_mask] [-h] [-r] [-t] [directory]" << std::endl
        << " -e event_mask: Event mask of possible values \"acdrnm\" - see the man page" << std::endl
        << " -h: This help" << std::endl
        << " -r: Return directly after first event" << std::endl
        << " -s: Be silent and just output the file name being the source of the event" << std::endl
        << " -t: Prefix each output with a timestamp" << std::endl
        << "The directory defaults to the current directory" << std::endl;
    std::exit(0);
}

void checkSystemCall(int returnCode) {
    if (returnCode == -1) {
        std::cerr << strerror(errno) << std::endl;
        std::exit(1);
    }
}

int main(int argc, char* argv[]) {
    settings.initialize();

    while (true) {
        char c = getopt(argc, argv, "e:hrst");
        if (c == -1) break;
        switch (c) {
            case 'e':
                std::cout << "EVENTS: " << optarg << std::endl;
                for (unsigned int i = 0; i < std::strlen(optarg); i++) {
                    switch (optarg[i]) {
                        case 'a':
                            settings.eventMask |= DN_ATTRIB;
                            break;
                        case 'c':
                            settings.eventMask |= DN_CREATE;

```

```

        break;
    case 'd':
        settings.eventMask |= DN_DELETE;
        break;
    case 'r':
        settings.eventMask |= DN_ACCESS;
        break;
    case 'n':
        settings.eventMask |= DN_RENAME;
        break;
    case 'm':
        settings.eventMask |= DN_MODIFY;
        break;
    default:
        std::cerr << "Unknown event attribute '" << optarg[i] << "' <<
std::endl;

        return 1;
    }
}
break;
case 'h':
    dieExplaining();
    break;
case 'r':
    settings.justDie = true;
    break;
case 's':
    settings.silent = true;
    break;
case 't':
    settings.timestamp = true;
    break;
};
}

if (settings.eventMask == DN_MULTISHOT) {
    settings.eventMask |= DN_ACCESS | DN_ATTRIB | DN_CREATE | DN_DELETE | DN_MODIFY | DN_RENAME;
}

std::string directory;
if (optind == argc) {
    directory = ".";
} else if (optind == argc - 1) {
    directory = argv[optind];
} else {
    dieExplaining();
}

struct sigaction act;
act.sa_sigaction = signalHandler;
act.sa_flags = SA_RESTART;
checkSystemCall(sigemptyset( &act.sa_mask ));
checkSystemCall(sigaction( SIGRTMIN + 1, &act, NULL ));

int fd = open( directory.c_str(), O_DIRECTORY | O_RDONLY );
checkSystemCall(fd);

struct dirent* directoryEntry;
DIR* dir = opendir(directory.c_str());
checkSystemCall((dir == NULL) ? -1 : 0);

while ((directoryEntry = readdir(dir)) != NULL) {
    struct stat buf;

    std::string fileName = std::string(directoryEntry->d_name);
    if (fileName == ".") continue;
    std::string fullPath = directory + "/" + fileName;

```

```

    checkSystemCall(stat(fullPath.c_str(), &buf));
    fileMap[directoryEntry->d_fileno] = File(buf, fileName, fullPath);
}
(void) closedir(dir);

fcntl(fd, F_SETSIG, SIGRTMIN + 1);
fcntl(fd, F_NOTIFY, settings.eventMask);

if (settings.justDie) {
    pause();
    return 0;
}

while (true) {
    pause();

    struct dirent* directoryEntry;
    DIR* dir = opendir(directory.c_str());
    if (dir == NULL) {
        std::cerr << strerror(errno) << std::endl;
        return 1;
    }
    while ((directoryEntry = readdir(dir)) != NULL) {
        struct stat buf;

        std::string fileName = std::string(directoryEntry->d_name);
        if (fileName == ".") continue;
        std::string fullPath = directory + "/" + fileName;

        checkSystemCall(stat(fullPath.c_str(), &buf));

        std::map<ino_t, File>::iterator it = fileMap.find(directoryEntry->d_fileno);
        if (it == fileMap.end()) {
            fileMap[directoryEntry->d_fileno] = File(buf, fileName, fullPath);
            timestamp(cout) << directoryEntry->d_name << (settings.silent ? "" : " created") <<
endl;
        } else {
            if (it->second.name != directoryEntry->d_name) {
                timestamp(cout) << it->second.name;
                if (not settings.silent) {
                    cout << " renamed[" << directoryEntry->d_name << "];"
                }
                cout << endl;
                it->second.name = directoryEntry->d_name;
                it->second.fullPath = directory + "/" + it->second.name;
            }
        }
    }
    (void) closedir(dir);

    for (std::map<ino_t, File>::iterator it = fileMap.begin(); it != fileMap.end(); it++) {
        File f = it->second;
        if (stat(f.fullPath.c_str(), &scratchStat) == -1) {
            if (errno == ENOENT) {
                timestamp(cout) << f.name << (settings.silent ? "" : " deleted") << endl;
                fileMap.erase(it);
                continue;
            }
        } else {
            it->second.update();
        }
    }
}
}

```