```
**********************************************************
   50779 Thu Jan 24 09:56:50 2019
new/usr/src/cmd/mv/mv.c
10134 mv needs smatch fixes
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

/*
 * Combined mv/cp/ln command:
 *      mv file1 file2
 *      mv dir1 dir2
 *      mv file1 ... filen dir1
 */
#include <sys/time.h>
#include <signal.h>
#include <locale.h>
#include <stdarg.h>
#include <sys/acl.h>
#include <libcmdutils.h>
#include <aclutils.h>
#include "getresponse.h"
```



```
#define FTYPE(A)        (A.st_mode)
#define FMODE(A)        (A.st_mode)
#define UID(A)          (A.st_uid)
#define GID(A)          (A.st_gid)
#define IDENTICAL(A, B) (A.st_dev == B.st_dev && A.st_ino == B.st_ino)
#define ISDIR(A)        ((A.st_mode & S_IFMT) == S_IFDIR)
#define ISDOOR(A)       ((A.st_mode & S_IFMT) == S_IFDOOR)
#define ISLNK(A)        ((A.st_mode & S_IFMT) == S_IFLNK)
#define ISREG(A)        (((A).st_mode & S_IFMT) == S_IFREG)
#define ISDEV(A)        ((A.st_mode & S_IFMT) == S_IFCHR || \
                        (A.st_mode & S_IFMT) == S_IFBLK || \
                        (A.st_mode & S_IFMT) == S_IFIFO)
#define ISSOCK(A)       ((A.st_mode & S_IFMT) == S_IFSOCK)

#define DELIM   '/'
#define EQ(x, y)        (strcmp(x, y) == 0)
#define FALSE   0
#define MODEBITS (S_ISUID|S_ISGID|S_ISVTX|S_IRWXU|S_IRWXG|S_IRWXO)
#define TRUE 1

static char             *dname(char *);
static int              lnkfil(char *, char *);
static int              cpymve(char *, char *);
static int              chkfiles(char *, char **);
static int              rcopy(char *, char *);
static int              chk_different(char *, char *);
static int              chg_time(char *, struct stat);
static int              chg_mode(char *, uid_t, gid_t, mode_t);
static int              copydir(char *, char *);
static int              copyspecial(char *);
static int              getrealpath(char *, char *);
static void             usage(void);
static void             Perror(char *);
static void             Perror2(char *, char *);
static int              use_stdin(void);
static int              copyattributes(char *, char *);
static int              copy_sysattr(char *, char *);
static tree_node_t      *create_tnode(dev_t, ino_t);

static struct stat      s1, s2, s3, s4;
static int              cpy = FALSE;
static int              mve = FALSE;
static int              lnk = FALSE;
static char             *cmd;
static int              silent = 0;
static int              fflg = 0;
static int              iflg = 0;
static int              pflg = 0;
static int              Rflg = 0;       /* recursive copy */
static int              rflg = 0;       /* recursive copy */
static int              sflg = 0;
static int              Hflg = 0;       /* follow cmd line arg symlink to dir */
static int              Lflg = 0;       /* follow symlinks */
static int              Pflg = 0;       /* do not follow symlinks */
static int              atflg = 0;
static int              attrsilent = 0;
static int              targetexists = 0;
static int              cmdarg;         /* command line argument */
static avl_tree_t       *stree = NULL;  /* source file inode search tree */
static acl_t            *s1acl;
static int              saflg = 0;      /* 'cp' extended system attr. */
static int              srcfd = -1;
static int              targfd = -1;
static int              sourcedirfd = -1;
static int              targetdirfd = -1;
```

```
static DIR              *srcdirp = NULL;
static int              srcattrfd = -1;
static int              targattrfd = -1;
static struct stat      attrdir;

/* Extended system attributes support */

static int open_source(char  *);
static int open_target_srctarg_attrdirs(char  *, char *);
static int open_attrdirp(char *);
static int traverse_attrfile(struct dirent *, char *, char *, int);
static void rewind_attrdir(DIR *);
static void close_all();


int
main(int argc, char *argv[])
{
        int c, i, r, errflg = 0;
        char target[PATH_MAX];
        int (*move)(char *, char *);

        /*
         * Determine command invoked (mv, cp, or ln)
         */

        if (cmd = strrchr(argv[0], '/'))
                ++cmd;
        else
                cmd = argv[0];

        /*
         * Set flags based on command.
         */

        (void) setlocale(LC_ALL, "");
#if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
#define TEXT_DOMAIN "SYS_TEST"  /* Use this only if it weren't */
#endif
        (void) textdomain(TEXT_DOMAIN);
        if (init_yes() < 0) {
                (void) fprintf(stderr, gettext(ERR_MSG_INIT_YES),
                    strerror(errno));
                exit(3);
        }

        if (EQ(cmd, "mv"))
                mve = TRUE;
        else if (EQ(cmd, "ln"))
                lnk = TRUE;
        else if (EQ(cmd, "cp"))
                cpy = TRUE;
        else {
                (void) fprintf(stderr,
                    gettext("Invalid command name (%s); expecting "
                    "mv, cp, or ln.\n"), cmd);
                exit(1);
        }

        /*
         * Check for options:
         *      cp [ -r|-R [-H|-L|-P]] [-afip@/] file1 [file2 ...] target
         *      cp [-afiprR@/] file1 [file2 ...] target
         *      ln [-f] [-n] [-s] file1 [file2 ...] target
         *      ln [-f] [-n] [-s] file1 [file2 ...]
         *      mv [-f|i] file1 [file2 ...] target
```



```
         *      mv [-f|i] dir1 target
         */

        if (cpy) {
                while ((c = getopt(argc, argv, "afHiLpPrR@/")) != EOF)
                        switch (c) {
                        case 'f':
                                fflg++;
                                break;
                        case 'i':
                                iflg++;
                                break;
                        case 'p':
                                pflg++;
#ifdef XPG4
                                attrsilent = 1;
                                atflg = 0;
                                saflg = 0;
#else
                                if (atflg == 0)
                                        attrsilent = 1;
#endif
                                break;
                        case 'H':
                                /*
                                 * If more than one of -H, -L, or -P are
                                 * specified, only the last option specified
                                 * determines the behavior.
                                 */
                                Lflg = Pflg = 0;
                                Hflg++;
                                break;
                        case 'L':
                                Hflg = Pflg = 0;
                                Lflg++;
                                break;
                        case 'P':
                                Lflg = Hflg = 0;
                                Pflg++;
                                break;
                        case 'R':
                                /*
                                 * The default behavior of cp -R|-r
                                 * when specified without -H|-L|-P
                                 * is -L.
                                 */
                                Rflg++;
                                /*FALLTHROUGH*/
                        case 'r':
                                rflg++;
                                break;
                        case 'a':
                                Lflg = Hflg = 0;
                                pflg++;
                                Pflg++;
                                Rflg++;
                                rflg++;
                                break;
                        case '@':
                                atflg++;
                                attrsilent = 0;
#ifdef XPG4
                                pflg = 0;
#endif
                                break;
                        case '/':
```

```
260                                 saflg++;
261                                 attrsilent = 0;
262 #ifdef XPG4
263                                 pflg = 0;
264 #endif
265                                 break;
266                         default:
267                                 errflg++;
268                         }

270                 /* -R or -r must be specified with -H, -L, or -P */
271                 if ((Hflg || Lflg || Pflg) && !(Rflg || rflg)) {
272                         errflg++;
273                 }

275         } else if (mve) {
276                 while ((c = getopt(argc, argv, "fis")) != EOF)
277                         switch (c) {
278                         case 'f':
279                                 silent++;
280 #ifdef XPG4
281                                 iflg = 0;
282 #endif
283                                 break;
284                         case 'i':
285                                 iflg++;
286 #ifdef XPG4
287                                 silent = 0;
288 #endif
289                                 break;
290                         default:
291                                 errflg++;
292                         }
293         } else { /* ln */
294                 while ((c = getopt(argc, argv, "fns")) != EOF)
295                         switch (c) {
296                         case 'f':
297                                 silent++;
298                                 break;
299                         case 'n':
300                                 /* silently ignored; this is the default */
301                                 break;
302                         case 's':
303                                 sflg++;
304                                 break;
305                         default:
306                                 errflg++;
307                         }
308         }

310         /*
311          * For BSD compatibility allow - to delimit the end of
312          * options for mv.
313          */
314         if (mve && optind < argc && (strcmp(argv[optind], "-") == 0))
315                 optind++;

317         /*
318          * Check for sufficient arguments
319          * or a usage error.
320          */

322         argc -= optind;
323         argv  = &argv[optind];

325         if ((argc < 2 && lnk != TRUE) || (argc < 1 && lnk == TRUE)) {
```



```
326                 (void) fprintf(stderr,
327                     gettext("%s: Insufficient arguments (%d)\n"),
328                     cmd, argc);
329                 usage();
330         }

332         if (errflg != 0)
333                 usage();

335         /*
336          * If there is more than a source and target,
337          * the last argument (the target) must be a directory
338          * which really exists.
339          */

341         if (argc > 2) {
342                 if (stat(argv[argc-1], &s2) < 0) {
343                         (void) fprintf(stderr,
344                             gettext("%s: %s not found\n"),
345                             cmd, argv[argc-1]);
346                         exit(2);
347                 }

349                 if (!ISDIR(s2)) {
350                         (void) fprintf(stderr,
351                             gettext("%s: Target %s must be a directory\n"),
352                             cmd, argv[argc-1]);
353                         usage();
354                 }
355         }

357         if (strlen(argv[argc-1]) >= PATH_MAX) {
358                 (void) fprintf(stderr,
359                     gettext("%s: Target %s file name length exceeds PATH_MAX"
360                     " %d\n"), cmd, argv[argc-1], PATH_MAX);
361                 exit(78);
362         }

364         if (argc == 1) {
365                 if (!lnk)
366                         usage();
367                 (void) strcpy(target, ".");
368         } else {
369                 (void) strcpy(target, argv[--argc]);
370         }

372         /*
373          * Perform a multiple argument mv|cp|ln by
374          * multiple invocations of cpymve() or lnkfil().
375          */
376         if (lnk)
377                 move = lnkfil;
378         else
379                 move = cpymve;

381         r = 0;
382         for (i = 0; i < argc; i++) {
383                 stree = NULL;
384                 cmdarg = 1;
385                 r += move(argv[i], target);
386         }

388         /*
389          * Show errors by nonzero exit code.
390          */
```

```
 392         return (r?2:0);
 393 }
______unchanged_portion_omitted_

1877 /* Copy extended system attributes from source to target */

1879 static int
1880 copy_sysattr(char *source, char *target)
1881 {
1882         struct dirent   *dp;
1883         nvlist_t        *response;
1884         int             error = 0;
1885         int             target_sa_support = 0;

1887         if (sysattr_support(source, _PC_SATTR_EXISTS) != 1)
1888                 return (0);

1890         if (open_source(source) != 0)
1891                 return (1);

1893         /*
1894          * Gets non default extended system attributes from the
1895          * source file to copy to the target. The target has
1896          * the defaults set when its created and thus  no need
1897          * to copy the defaults.
1898          */
1899         response = sysattr_list(cmd, srcfd, source);

1901         if (sysattr_support(target, _PC_SATTR_ENABLED) != 1) {
1902                 if (response != NULL) {
1903                         (void) fprintf(stderr,
1904                             gettext(
1905                             "%s: cannot preserve extended system "
1906                             "attribute, operation not supported on file"
1907                             " %s\n"), cmd, target);
1908                         error++;
1909                         goto out;
1910                 }
1911         } else {
1912                 target_sa_support = 1;
1913         }

1915         if (target_sa_support) {
1916                 if (srcdirp == NULL) {
1917                         if (open_target_srctarg_attrdirs(source,
1918                             target) !=  0) {
1919                                 error++;
1920                                 goto out;
1921                         }
1922                         if (open_attrdirp(source) != 0) {
1923                                 error++;
1924                                 goto out;
1925                         }
1926                 } else {
1927                         rewind_attrdir(srcdirp);
1928                 }
1929                 while ((dp = readdir(srcdirp)) != NULL) {
1930                         nvlist_t        *res;
1931                         int             ret;

1933                         if ((ret = traverse_attrfile(dp, source, target,
1934                             0)) == -1)
1935                                 continue;
1936                         else if (ret > 0) {
1937                                 ++error;
1938                                 goto out;
```



```
1939                         }
1940                         /*
1941                          * Gets non default extended system attributes from the
1942                          * attribute file to copy to the target. The target has
1943                          * the defaults set when its created and thus  no need
1944                          * to copy the defaults.
1945                          */
1942                         if (dp->d_name != NULL) {
1946                         res = sysattr_list(cmd, srcattrfd, dp->d_name);
1947                         if (res == NULL)
1948                                 goto next;

1950                         /*
1951                          * Copy non default extended system attributes of named
1952                          * attribute file.
1953                          */
1954                         if (fsetattr(targattrfd,
1955                             XATTR_VIEW_READWRITE, res) != 0) {
1956                                 ++error;
1957                                 (void) fprintf(stderr, gettext("%s: "
1958                                     "Failed to copy extended system "
1959                                     "attributes from attribute file "
1960                                     "%s of %s to %s\n"), cmd,
1961                                     dp->d_name, source, target);
1962                         }

1960                         }
1964 next:
1965                         if (srcattrfd != -1)
1966                                 (void) close(srcattrfd);
1967                         if (targattrfd != -1)
1968                                 (void) close(targattrfd);
1969                         srcattrfd = targattrfd = -1;
1970                         nvlist_free(res);
1971                 }
1972         }
1973         /* Copy source file non default extended system attributes to target */
1974         if (target_sa_support && (response != NULL) &&
1975             (fsetattr(targfd, XATTR_VIEW_READWRITE, response)) != 0) {
1976                 ++error;
1977                 (void) fprintf(stderr, gettext("%s: Failed to "
1978                     "copy extended system attributes from "
1979                     "%s to %s\n"), cmd, source, target);
1980         }
1981 out:
1982         nvlist_free(response);
1983         close_all();
1984         return (error == 0 ? 0 : 1);
1985 }
______unchanged_portion_omitted_
```