```
**********************************************************
   71712 Sat Mar  7 16:23:21 2015
new/usr/src/cmd/ls/ls.c
5613 odd ls -U behaviour if output is not a terminal
**********************************************************
```

```
/*
 * 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 (c) 1988, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2012, Joyent, Inc. All rights reserved.
 * Copyright 2015 Gary Mills
 */

/*
 * Copyright 2009 Jason King.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*      Copyright (c) 1987, 1988 Microsoft Corporation  */
/*        All Rights Reserved   */

/*
 * List files or directories
 */

#include <sys/param.h>
#include <sys/types.h>
#include <sys/mkdev.h>
#include <sys/stat.h>
#include <sys/acl.h>

#include <wchar.h>
#include <stdio.h>
#include <ctype.h>
#include <dirent.h>
#include <string.h>
#include <locale.h>
#include <curses.h>
#include <term.h>
#include <termios.h>
#include <stdlib.h>
#include <widec.h>
#include <locale.h>
#include <wctype.h>
```



```
#include <pwd.h>
#include <grp.h>
#include <limits.h>
#include <fcntl.h>
#include <unistd.h>
#include <libgen.h>
#include <errno.h>
#include <aclutils.h>
#include <libnvpair.h>
#include <libcmdutils.h>
#include <attr.h>
#include <getopt.h>
#include <inttypes.h>

#ifndef STANDALONE
#define TERMINFO
#endif

/*
 * -DNOTERMINFO can be defined on the cc command line to prevent
 * the use of terminfo.  This should be done on systems not having
 * the terminfo feature(pre 6.0 systems ?).
 * As a result, columnar listings assume 80 columns for output,
 * unless told otherwise via the COLUMNS environment variable.
 */
#ifdef NOTERMINFO
#undef TERMINFO
#endif

#include <term.h>

#define BFSIZE  16
/* this bit equals 1 in lflags of structure lbuf if *namep is to be used */
#define ISARG   0100000

/*
 * this flag has been added to manipulate the display of S instead of 'l' when
 * the file is not a regular file and when group execution bit is off
 */
#define LS_NOTREG       010000


/*
 * Date and time formats
 *
 * b --- abbreviated month name
 * e --- day number
 * Y --- year in the form ccyy
 * H --- hour(24-hour version)
 * M --- minute
 * F --- yyyy-mm-dd
 * T --- hh:mm:ss
 * z --- time zone as hours displacement from UTC
 * note that %F and %z are from the ISO C99 standard and are
 * not present in older C libraries
 */
#define FORMAT_OLD      " %b %e  %Y "
#define FORMAT_NEW      " %b %e %H:%M "
#define FORMAT_LONG     " %b %e %T %Y "
#define FORMAT_ISO_FULL " %%F %%T.%.09ld %%z "
#define FORMAT_ISO_LONG " %F %R "
#define FORMAT_ISO_NEW  " %m-%d %H:%M "
#define FORMAT_ISO_OLD  " %F "

#undef BUFSIZ
#define BUFSIZ 4096
```

```
 128 #define NUMBER_WIDTH 40
 129 #define FMTSIZE 50

 131 struct ditem {
 132         dev_t   dev;                    /* directory items device number */
 133         ino_t   ino;                    /* directory items inode number */
 134         struct ditem *parent;           /* dir items ptr to its parent's info */
 135 };
_____unchanged_portion_omitted_

1248 /*
1249  * print one output entry;
1250  * if uid/gid is not found in the appropriate
1251  * file(passwd/group), then print uid/gid instead of
1252  * user/group name;
1253  */
1254 static void
1255 pentry(struct lbuf *ap)
1256 {
1257         struct lbuf *p;
1258         numbuf_t hbuf;
1259         char *dmark = "";       /* Used if -p or -F option active */
1260         char *cp;
1261         char *str;

1263         if (noflist) {
1264                 (void) printf("%s\n", (ap->lflags & ISARG) ? ap->ln.namep :
1265                     ap->ln.lname);
1263                 (void) printf("%s\n", ap->ln.lname);
1266                 return;
1267         }

1269         p = ap;
1270         column();
1271         if (iflg) {
1272                 if (mflg && !lflg)
1273                         curcol += printf("%llu ", (long long)p->lnum);
1274                 else
1275                         curcol += printf("%10llu ", (long long)p->lnum);
1276         }
1277         if (sflg) {
1278                 curcol += printf((mflg && !lflg) ? "%lld " :
1279                     (p->lblocks < 10000) ? "%4lld " : "%lld ",
1280                     (p->ltype != 'b' && p->ltype != 'c') ?
1281                     p->lblocks : 0LL);
1282         }
1283         if (lflg) {
1284                 (void) putchar(p->ltype);
1285                 curcol++;
1286                 pmode(p->lflags);

1288                 /* ACL: additional access mode flag */
1289                 (void) putchar(p->acl);
1290                 curcol++;

1292                 curcol += printf("%3lu ", (ulong_t)p->lnl);
1293                 if (oflg) {
1294                         if (!nflg) {
1295                                 cp = getname(p->luid);
1296                                 curcol += printf("%-8s ", cp);
1297                         } else
1298                                 curcol += printf("%-8lu ", (ulong_t)p->luid);
1299                 }
1300                 if (gflg) {
1301                         if (!nflg) {
1302                                 cp = getgroup(p->lgid);
```



```
1303                                 curcol += printf("%-8s ", cp);
1304                         } else
1305                                 curcol += printf("%-8lu ", (ulong_t)p->lgid);
1306                 }
1307                 if (p->ltype == 'b' || p->ltype == 'c') {
1308                         curcol += printf("%3u, %2u",
1309                             (uint_t)major((dev_t)p->lsize),
1310                             (uint_t)minor((dev_t)p->lsize));
1311                 } else if (hflg && (p->lsize >= hscale)) {
1312                         curcol += printf("%7s",
1313                             number_to_scaled_string(hbuf, p->lsize, hscale));
1314                 } else {
1315                         uint64_t bsize = p->lsize / block_size;

1317                         /*
1318                          * Round up only when using blocks > 1 byte, otherwise
1319                          * 'normal' sizes display 1 byte too large.
1320                          */
1321                         if (p->lsize % block_size != 0)
1322                                 bsize++;

1324                         curcol += printf("%7" PRIu64, bsize);
1325                 }
1326                 format_time(p->lmtime.tv_sec, p->lmtime.tv_nsec);
1327                 /* format extended system attribute time */
1328                 if (tmflg && crtm)
1329                         format_attrtime(p);

1331                 curcol += printf("%s", time_buf);

1333         }
1334         /*
1335          * prevent both "->" and trailing marks
1336          * from appearing
1337          */

1339         if (pflg && p->ltype == 'd')
1340                 dmark = "/";

1342         if (Fflg && !(lflg && p->flinkto)) {
1343                 if (p->ltype == 'd')
1344                         dmark = "/";
1345                 else if (p->ltype == 'D')
1346                         dmark = ">";
1347                 else if (p->ltype == 'p')
1348                         dmark = "|";
1349                 else if (p->ltype == 'l')
1350                         dmark = "@";
1351                 else if (p->ltype == 's')
1352                         dmark = "=";
1353                 else if (!file_typeflg &&
1354                     (p->lflags & (S_IXUSR|S_IXGRP|S_IXOTH)))
1355                         dmark = "*";
1356                 else
1357                         dmark = "";
1358         }

1360         if (colorflg)
1361                 ls_start_color(p->color);

1363         if (p->lflags & ISARG)
1364                 str = p->ln.namep;
1365         else
1366                 str = p->ln.lname;

1368         if (qflg || bflg) {
```

```
1369                csi_pprintf((unsigned char *)str);

1371                if (lflg && p->flinkto) {
1372                        if (colorflg)
1373                                ls_end_color();
1374                        csi_pprintf((unsigned char *)" -> ");
1375                        if (colorflg)
1376                                ls_start_color(p->link_color);
1377                        csi_pprintf((unsigned char *)p->flinkto);
1378                } else {
1379                        csi_pprintf((unsigned char *)dmark);
1380                }
1381        } else {
1382                (void) printf("%s", str);
1383                curcol += strcol((unsigned char *)str);

1385                if (lflg && p->flinkto) {
1386                        if (colorflg)
1387                                ls_end_color();
1388                        str = " -> ";
1389                        (void) printf("%s", str);
1390                        curcol += strcol((unsigned char *)str);
1391                        if (colorflg)
1392                                ls_start_color(p->link_color);
1393                        (void) printf("%s", p->flinkto);
1394                        curcol += strcol((unsigned char *)p->flinkto);
1395                } else {
1396                        (void) printf("%s", dmark);
1397                        curcol += strcol((unsigned char *)dmark);
1398                }
1399        }

1401        if (colorflg)
1402                ls_end_color();

1404        /* Display extended system attributes */
1405        if (saflg) {
1406                int i;

1408                new_line();
1409                (void) printf(" \t{");
1410                if (p->exttr != NULL) {
1411                        int k = 0;
1412                        for (i = 0; i < sacnt; i++) {
1413                                if (p->exttr[i].name != NULL)
1414                                        k++;
1415                        }
1416                        for (i = 0; i < sacnt; i++) {
1417                                if (p->exttr[i].name != NULL) {
1418                                        (void) printf("%s", p->exttr[i].name);
1419                                        k--;
1420                                        if (vopt && (k != 0))
1421                                                (void) printf(",");
1422                                }
1423                        }
1424                }
1425                (void) printf("}\n");
1426        }
1427        /* Display file timestamps and extended system attribute timestamps */
1428        if (tmflg && alltm) {
1429                new_line();
1430                print_time(p);
1431                new_line();
1432        }
1433        if (vflg) {
1434                new_line();
```



```
1435                if (p->aclp) {
1436                        acl_printacl(p->aclp, num_cols, Vflg);
1437                }
1438        }
1439        /* Free extended system attribute lists */
1440        if (saflg || tmflg)
1441                free_sysattr(p);
1442 }
```

_____***unchanged_portion_omitted_***

```
1760 /*
1761  * get status of file and recomputes tblocks;
1762  * argfl = 1 if file is a name in ls-command and = 0
1763  * for filename in a directory whose name is an
1764  * argument in the command;
1765  * stores a pointer in flist[nfiles] and
1766  * returns that pointer;
1767  * returns NULL if failed;
1768  */
1769 static struct lbuf *
1770 gstat(char *file, int argfl, struct ditem *myparent)
1771 {
1772        struct stat statb, statb1;
1773        struct lbuf *rep;
1774        char buf[BUFSIZ];
1775        ssize_t cc;
1776        int (*statf)() = ((Lflg) || (Hflg && argfl)) ? stat : lstat;
1777        int aclcnt;
1778        int error;
1779        aclent_t *tp;
1780        o_mode_t groupperm, mask;
1781        int grouppermfound, maskfound;

1783        if (nomocore)
1784                return (NULL);

1786        if (nfiles >= maxfils) {
1787                /*
1788                 * all flist/lbuf pair assigned files, time to get some
1789                 * more space
1790                 */
1791                maxfils += quantn;
1792                if (((flist = realloc(flist,
1793                    maxfils * sizeof (struct lbuf *))) == NULL) ||
1794                    ((nxtlbf = malloc(quantn *
1795                    sizeof (struct lbuf))) == NULL)) {
1796                        perror("ls");
1797                        nomocore = 1;
1798                        return (NULL);
1799                }
1800        }

1802        /*
1803         * nfiles is reset to nargs for each directory
1804         * that is given as an argument maxn is checked
1805         * to prevent the assignment of an lbuf to a flist entry
1806         * that already has one assigned.
1807         */
1808        if (nfiles >= maxn) {
1809                rep = nxtlbf++;
1810                flist[nfiles++] = rep;
1811                maxn = nfiles;
1812        } else {
1813                rep = flist[nfiles++];
1814        }
```

```
1816         /* Clear the lbuf */
1817         (void) memset((void *) rep, 0, sizeof (struct lbuf));

1819         /*
1820          * When noflist is set, none of the extra information about the dirent
1821          * will be printed, so omit remaining initialization of this lbuf
1822          * as well as the  stat(2) call.
1816          * will be printed, so omit initialization of this lbuf as well as the
1817          * stat(2) call.
1823          */
1824         if (!argfl && noflist)
1825                 return (rep);

1827         /* Initialize non-zero members */
1822         /* Initialize */

1824         rep->lflags = (mode_t)0;
1825         rep->flinkto = NULL;
1826         rep->cycle = 0;
1829         rep->lat.tv_sec = time(NULL);
1828         rep->lat.tv_nsec = 0;
1830         rep->lct.tv_sec = time(NULL);
1830         rep->lct.tv_nsec = 0;
1831         rep->lmt.tv_sec = time(NULL);
1832         rep->lmt.tv_nsec = 0;
1833         rep->aclp = NULL;
1834         rep->exttr = NULL;
1835         rep->extm = NULL;
1836         rep->color = NULL;
1837         rep->link_color = NULL;

1833         if (argfl || statreq) {
1834                 int doacl;

1836                 if (lflg)
1837                         doacl = 1;
1838                 else
1839                         doacl = 0;

1841                 if ((*statf)(file, &statb) < 0) {
1842                         if (argfl || errno != ENOENT ||
1843                             (lflg && lstat(file, &statb) == 0)) {
1844                                 /*
1845                                  * Avoid race between readdir and lstat.
1846                                  * Print error message in case of dangling link.
1847                                  */
1848                                 perror(file);
1849                                 err = 2;
1850                         }
1851                         nfiles--;
1852                         return (NULL);
1853                 }

1855                 /*
1856                  * If -H was specified, and the file linked to was
1857                  * not a directory, then we need to get the info
1858                  * for the symlink itself.
1859                  */
1860                 if ((Hflg) && (argfl) &&
1861                     ((statb.st_mode & S_IFMT) != S_IFDIR)) {
1862                         if (lstat(file, &statb) < 0) {
1863                                 perror(file);
1864                                 err = 2;
1865                         }
1866                 }
```



```
1868                 rep->lnum = statb.st_ino;
1869                 rep->lsize = statb.st_size;
1870                 rep->lblocks = statb.st_blocks;
1871                 if (colorflg)
1872                         rep->color = ls_color_find(file, statb.st_mode);

1874                 switch (statb.st_mode & S_IFMT) {
1875                 case S_IFDIR:
1876                         rep->ltype = 'd';
1877                         if (Rflg) {
1878                                 record_ancestry(file, &statb, rep,
1879                                     argfl, myparent);
1880                         }
1881                         break;
1882                 case S_IFBLK:
1883                         rep->ltype = 'b';
1884                         rep->lsize = (off_t)statb.st_rdev;
1885                         break;
1886                 case S_IFCHR:
1887                         rep->ltype = 'c';
1888                         rep->lsize = (off_t)statb.st_rdev;
1889                         break;
1890                 case S_IFIFO:
1891                         rep->ltype = 'p';
1892                         break;
1893                 case S_IFSOCK:
1894                         rep->ltype = 's';
1895                         rep->lsize = 0;
1896                         break;
1897                 case S_IFLNK:
1898                         /* symbolic links may not have ACLs, so elide acl() */
1899                         if ((Lflg == 0) || (Hflg == 0) ||
1900                             ((Hflg) && (!argfl))) {
1901                                 doacl = 0;
1902                         }
1903                         rep->ltype = 'l';
1904                         if (lflg || colorflg) {
1905                                 cc = readlink(file, buf, BUFSIZ);
1906                                 if (cc < 0)
1907                                         break;

1909                                 /*
1910                                  * follow the symbolic link
1911                                  * to generate the appropriate
1912                                  * Fflg marker for the object
1913                                  * eg, /bin -> /sym/bin/
1914                                  */
1915                                 error = 0;
1916                                 if (Fflg || pflg || colorflg)
1917                                         error = stat(file, &statb1);

1919                                 if (colorflg) {
1920                                         if (error >= 0)
1921                                                 rep->link_color =
1922                                                     ls_color_find(file,
1923                                                     statb1.st_mode);
1924                                         else
1925                                                 rep->link_color =
1926                                                     lsc_orphan;
1927                                 }

1929                                 if ((Fflg || pflg) && error >= 0) {
1930                                         switch (statb1.st_mode & S_IFMT) {
1931                                         case S_IFDIR:
1932                                                 buf[cc++] = '/';
1933                                                 break;
```

```
1934                                    case S_IFSOCK:
1935                                            buf[cc++] = '=';
1936                                            break;
1937                                    case S_IFDOOR:
1938                                            buf[cc++] = '>';
1939                                            break;
1940                                    case S_IFIFO:
1941                                            buf[cc++] = '|';
1942                                            break;
1943                                    default:
1944                                            if ((statb1.st_mode & ~S_IFMT) &
1945                                                (S_IXUSR|S_IXGRP|S_IXOTH))
1946                                                    buf[cc++] = '*';
1947                                            break;
1948                                    }
1949                            }
1950                            buf[cc] = '\0';
1951                            rep->flinkto = strdup(buf);
1952                            if (rep->flinkto == NULL) {
1953                                    perror("ls");
1954                                    nomocore = 1;
1955                                    return (NULL);
1956                            }
1957                            break;
1958                    }

1960                    /*
1961                     * ls /sym behaves differently from ls /sym/
1962                     * when /sym is a symbolic link. This is fixed
1963                     * when explicit arguments are specified.
1964                     */

1966 #ifdef XPG6
1967                     /* Do not follow a symlink when -F is specified */
1968                     if ((!argfl) || (argfl && Fflg) ||
1969                         (stat(file, &statb1) < 0))
1970 #else
1971                     /* Follow a symlink when -F is specified */
1972                     if (!argfl || stat(file, &statb1) < 0)
1973 #endif /* XPG6 */
1974                             break;
1975                     if ((statb1.st_mode & S_IFMT) == S_IFDIR) {
1976                             statb = statb1;
1977                             rep->ltype = 'd';
1978                             rep->lsize = statb1.st_size;
1979                             if (Rflg) {
1980                                     record_ancestry(file, &statb, rep,
1981                                         argfl, myparent);
1982                             }
1983                     }
1984                     break;
1985             case S_IFDOOR:
1986                     rep->ltype = 'D';
1987                     break;
1988             case S_IFREG:
1989                     rep->ltype = '-';
1990                     break;
1991             case S_IFPORT:
1992                     rep->ltype = 'P';
1993                     break;
1994             default:
1995                     rep->ltype = '?';
1996                     break;
1997             }
1998             rep->lflags = statb.st_mode & ~S_IFMT;
```



```
2000             if (!S_ISREG(statb.st_mode))
2001                     rep->lflags |= LS_NOTREG;

2003             rep->luid = statb.st_uid;
2004             rep->lgid = statb.st_gid;
2005             rep->lnl = statb.st_nlink;
2006             if (uflg || (tmflg && atm))
2007                     rep->lmtime = statb.st_atim;
2008             else if (cflg || (tmflg && ctm))
2009                     rep->lmtime = statb.st_ctim;
2010             else
2011                     rep->lmtime = statb.st_mtim;
2012             rep->lat = statb.st_atim;
2013             rep->lct = statb.st_ctim;
2014             rep->lmt = statb.st_mtim;

2016             /* ACL: check acl entries count */
2017             if (doacl) {

2019                     error = acl_get(file, 0, &rep->aclp);
2020                     if (error) {
2021                             (void) fprintf(stderr,
2022                                 gettext("ls: can't read ACL on %s: %s\n"),
2023                                 file, acl_strerror(error));
2024                             rep->acl = ' ';
2025                             acl_err++;
2026                             return (rep);
2027                     }

2029                     rep->acl = ' ';

2031                     if (rep->aclp &&
2032                         ((acl_flags(rep->aclp) & ACL_IS_TRIVIAL) == 0)) {
2033                             rep->acl = '+';
2034                             /*
2035                              * Special handling for ufs aka aclent_t ACL's
2036                              */
2037                             if (acl_type(rep->aclp) == ACLENT_T) {
2038                                     /*
2039                                      * For files with non-trivial acls, the
2040                                      * effective group permissions are the
2041                                      * intersection of the GROUP_OBJ value
2042                                      * and the CLASS_OBJ (acl mask) value.
2043                                      * Determine both the GROUP_OBJ and
2044                                      * CLASS_OBJ for this file and insert
2045                                      * the logical AND of those two values
2046                                      * in the group permissions field
2047                                      * of the lflags value for this file.
2048                                      */

2050                                     /*
2051                                      * Until found in acl list, assume
2052                                      * maximum permissions for both group
2053                                      * a nd mask.  (Just in case the acl
2054                                      * lacks either value for some reason.)
2055                                      */
2056                                     groupperm = 07;
2057                                     mask = 07;
2058                                     grouppermfound = 0;
2059                                     maskfound = 0;
2060                                     aclcnt = acl_cnt(rep->aclp);
2061                                     for (tp =
2062                                         (aclent_t *)acl_data(rep->aclp);
2063                                         aclcnt--; tp++) {
2064                                             if (tp->a_type == GROUP_OBJ) {
2065                                                     groupperm = tp->a_perm;
```

```
2066                                                    grouppermfound = 1;
2067                                                    continue;
2068                                            }
2069                                            if (tp->a_type == CLASS_OBJ) {
2070                                                    mask = tp->a_perm;
2071                                                    maskfound = 1;
2072                                            }
2073                                            if (grouppermfound && maskfound)
2074                                                    break;
2075                                    }


2078                                    /* reset all the group bits */
2079                                    rep->lflags &= ~S_IRWXG;

2081                                    /*
2082                                     * Now set them to the logical AND of
2083                                     * the GROUP_OBJ permissions and the
2084                                     * acl mask.
2085                                     */

2087                                    rep->lflags |= (groupperm & mask) << 3;

2089                            } else if (acl_type(rep->aclp) == ACE_T) {
2090                                    int mode;
2091                                    mode = grp_mask_to_mode(rep);
2092                                    rep->lflags &= ~S_IRWXG;
2093                                    rep->lflags |= mode;
2094                            }
2095                    }

2097                    if (!vflg && !Vflg && rep->aclp) {
2098                            acl_free(rep->aclp);
2099                            rep->aclp = NULL;
2100                    }

2102                    if (atflg && pathconf(file, _PC_XATTR_EXISTS) == 1)
2103                            rep->acl = '@';

2105            } else
2106                    rep->acl = ' ';

2108            /* mask ISARG and other file-type bits */

2110            if (rep->ltype != 'b' && rep->ltype != 'c')
2111                    tblocks += rep->lblocks;

2113            /* Get extended system attributes */

2115            if ((saflg || (tmflg && crtm) || (tmflg && alltm)) &&
2116                (sysattr_support(file, _PC_SATTR_EXISTS) == 1)) {
2117                    int i;

2119                    sacnt = attr_count();
2120                    /*
2121                     * Allocate 'sacnt' size array to hold extended
2122                     * system attribute name (verbose) or respective
2123                     * symbol represenation (compact).
2124                     */
2125                    rep->exttr = xmalloc(sacnt * sizeof (struct attrb),
2126                        rep);

2128                    /* initialize boolean attribute list */
2129                    for (i = 0; i < sacnt; i++)
2130                            rep->exttr[i].name = NULL;
2131                    if (get_sysxattr(file, rep) != 0) {
```



```
2132                                    (void) fprintf(stderr,
2133                                        gettext("ls:Failed to retrieve "
2134                                        "extended system attribute from "
2135                                        "%s\n"), file);
2136                                    rep->exttr[0].name = xmalloc(2, rep);
2137                                    (void) strlcpy(rep->exttr[0].name, "?", 2);
2138                            }
2139                    }
2140            }
2141            return (rep);
2142 }
```

_____***unchanged_portion_omitted_***