

new/usr/src/cmd/getfacl/Makefile

1

\*\*\*\*\*

1061 Sat Feb 8 11:08:22 2020

new/usr/src/cmd/getfacl/Makefile

12288 getfacl and setfacl could stand improvement

\*\*\*\*\*

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License, Version 1.0 only
6 # (the "License"). You may not use this file except in compliance
7 # with the License.
8 #
9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 # or http://www.opensolaris.org/os/licensing.
11 # See the License for the specific language governing permissions
12 # and limitations under the License.
13 #
14 # When distributing Covered Code, include this CDDL HEADER in each
15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 # If applicable, add the following below this CDDL HEADER, with the
17 # fields enclosed by brackets "[]" replaced with your own identifying
18 # information: Portions Copyright [yyyy] [name of copyright owner]
19 #
20 # CDDL HEADER END
21 #
22 #
23 # Copyright (c) 1993,2001 by Sun Microsystems, Inc.
24 # All rights reserved.
25 #
26 # Copyright 2020 Peter Tribble.
27 #

29 PROG= getfacl

31 include ../Makefile.cmd

31 CERRWARN += _gcc=-Wno-unused-variable
32 CERRWARN += $(CNOWARN_UNINIT)

33 .KEEP_STATE:

35 all: $(PROG)

37 install: all $(ROOTPROG)

39 clean:

42 lint: lint_PROG

41 include ../Makefile.targ
```

```

*****
7056 Sat Feb 8 11:08:22 2020
new/usr/src/cmd/getfacl/getfacl.c
12288 getfacl and setfacl could stand improvement
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2007 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 * Copyright 2020 Peter Tribble.
25 */

26 #pragma ident "%Z%M% %I% %E% SMI"

28 #ifndef lint
29 static char sccsid[] = "%Z%M% %I% %E% SMI";
30 #endif

27 /*
28 * getfacl [-ad] file ...
29 * This command displays discretionary information for a file or files.
30 * display format:
31 * # file: filename
32 * # owner: uid
33 * # group: gid
34 * user::perm
35 * user:uid:perm
36 * group::perm
37 * group:gid:perm
38 * mask:perm
39 * other:perm
40 * default:user::perm
41 * default:user:uid:perm
42 * default:group::perm
43 * default:group:gid:perm
44 * default:mask:perm
45 * default:other:perm
46 */

48 #include <stdlib.h>
49 #include <stdio.h>
50 #include <pwd.h>
51 #include <grp.h>
52 #include <locale.h>
53 #include <sys/acl.h>
54 #include <errno.h>

```

```

56 static char *pruname(uid_t);
57 static char *prgname(gid_t);
58 static char *display(int);
59 static void usage();

62 int
63 main(int argc, char *argv[])
64 {
65     int c;
66     int aflag = 0;
67     int dflag = 0;
68     int errflag = 0;
69     int savecnt;
70     int aclcnt;
71     int mask = 0;
72     int *aclp;
73     int *tp;
74     char *permp;

76     (void) setlocale(LC_ALL, "");
77     (void) textdomain(TEXT_DOMAIN);

79     if (argc < 2)
80         usage();

82     while ((c = getopt(argc, argv, "ad")) != EOF) {
83         switch (c) {
84             case 'a':
85                 aflag++;
86                 break;
87             case 'd':
88                 dflag++;
89                 break;
90             case '?':
91                 errflag++;
92                 break;
93         }
94     }
95     if (errflag)
96         usage();

98     if (optind >= argc)
99         usage();

101     for (; optind < argc; optind++) {
102         register char *filep;

104         filep = argv[optind];

106         /* Get ACL info of the files */
107         errno = 0;
108         if ((aclcnt = acl(filep, GETACLNT, 0, NULL)) < 0) {
109             if (errno == ENOSYS) {
110                 (void) fprintf(stderr,
111                     gettext("File system doesn't support "
112                         "aclent_t style ACL's.\n"
113                         "See acl(5) for more information on "
114                         "POSIX-draft ACL support.\n"));
115                 (void) fprintf(stderr,
116                     "Solaris ACL support.\n");
117                 exit(2);
118             }
119             perror(filep);
120             exit(2);
121         }
122     }

```

```

120     if (aclcnt < MIN_ACL_ENTRIES) {
121         (void) fprintf(stderr,
122             gettext("%d: acl count too small from %s\n"),
123             aclcnt, filep);
124         exit(2);
125     }
127     if ((aclp = (aclent_t *)malloc(sizeof (aclent_t) * aclcnt))
128         == NULL) {
129         (void) fprintf(stderr,
130             gettext("Insufficient memory\n"));
131         exit(1);
132     }
134     errno = 0;
135     if (acl(filep, GETACL, aclcnt, aclp) < 0) {
136         perror(filep);
137         exit(2);
138     }
140     /* display ACL: assume it is sorted. */
141     (void) printf("\n# file: %s\n", filep);
142     savecnt = aclcnt;
143     for (tp = aclp; aclcnt--; tp++) {
144         if (tp->a_type == USER_OBJ)
145             (void) printf("# owner: %s\n",
146                 pruname(tp->a_id));
147         if (tp->a_type == GROUP_OBJ)
148             (void) printf("# group: %s\n",
149                 pruname(tp->a_id));
150         if (tp->a_type == CLASS_OBJ)
151             mask = tp->a_perm;
152     }
153     aclcnt = savecnt;
154     for (tp = aclp; aclcnt--; tp++) {
155         switch (tp->a_type) {
156             case USER:
157                 if (!dflag) {
158                     permp = display(tp->a_perm);
159                     (void) printf("user:%s:%s\t\t",
160                         pruname(tp->a_id), permp);
161                     free(permp);
162                     permp = display(tp->a_perm & mask);
163                     (void) printf(
164                         "#effective:%s\n", permp);
165                     free(permp);
166                 }
167                 break;
168             case USER_OBJ:
169                 if (!dflag) {
170                     /* no need to display uid */
171                     permp = display(tp->a_perm);
172                     (void) printf("user::%s\n", permp);
173                     free(permp);
174                 }
175                 break;
176             case GROUP:
177                 if (!dflag) {
178                     permp = display(tp->a_perm);
179                     (void) printf("group:%s:%s\t\t",
180                         pruname(tp->a_id), permp);
181                     free(permp);
182                     permp = display(tp->a_perm & mask);
183                     (void) printf(
184                         "#effective:%s\n", permp);
185                     free(permp);

```

```

186     }
187     break;
188     case GROUP_OBJ:
189         if (!dflag) {
190             permp = display(tp->a_perm);
191             (void) printf("group::%s\t\t", permp);
192             free(permp);
193             permp = display(tp->a_perm & mask);
194             (void) printf(
195                 "#effective:%s\n", permp);
196             free(permp);
197         }
198         break;
199     case CLASS_OBJ:
200         if (!dflag) {
201             permp = display(tp->a_perm);
202             (void) printf("mask:%s\n", permp);
203             free(permp);
204         }
205         break;
206     case OTHER_OBJ:
207         if (!dflag) {
208             permp = display(tp->a_perm);
209             (void) printf("other:%s\n", permp);
210             free(permp);
211         }
212         break;
213     case DEF_USER:
214         if (!aflag) {
215             permp = display(tp->a_perm);
216             (void) printf("default:user:%s:%s\n",
217                 pruname(tp->a_id), permp);
218             free(permp);
219         }
220         break;
221     case DEF_USER_OBJ:
222         if (!aflag) {
223             permp = display(tp->a_perm);
224             (void) printf("default:user::%s\n",
225                 permp);
226             free(permp);
227         }
228         break;
229     case DEF_GROUP:
230         if (!aflag) {
231             permp = display(tp->a_perm);
232             (void) printf("default:group:%s:%s\n",
233                 pruname(tp->a_id), permp);
234             free(permp);
235         }
236         break;
237     case DEF_GROUP_OBJ:
238         if (!aflag) {
239             permp = display(tp->a_perm);
240             (void) printf("default:group::%s\n",
241                 permp);
242             free(permp);
243         }
244         break;
245     case DEF_CLASS_OBJ:
246         if (!aflag) {
247             permp = display(tp->a_perm);
248             (void) printf("default:mask:%s\n",
249                 permp);
250             free(permp);
251         }

```

```
252         break;
253     case DEF_OTHER_OBJ:
254         if (!aflag) {
255             permp = display(tp->a_perm);
256             (void) printf("default:other:%s\n",
257                 permp);
258             free(permp);
259         }
260         break;
261     default:
262         (void) fprintf(stderr,
263             gettext("unrecognized entry\n"));
264         break;
265     }
266     free(ac1p);
267 }
268 return (0);
269 }
270 }
unchanged_portion_omitted
```

new/usr/src/cmd/setfacl/Makefile

1

\*\*\*\*\*

1084 Sat Feb 8 11:08:23 2020

new/usr/src/cmd/setfacl/Makefile

12288 getfacl and setfacl could stand improvement

\*\*\*\*\*

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License, Version 1.0 only
6 # (the "License"). You may not use this file except in compliance
7 # with the License.
8 #
9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 # or http://www.opensolaris.org/os/licensing.
11 # See the License for the specific language governing permissions
12 # and limitations under the License.
13 #
14 # When distributing Covered Code, include this CDDL HEADER in each
15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 # If applicable, add the following below this CDDL HEADER, with the
17 # fields enclosed by brackets "[]" replaced with your own identifying
18 # information: Portions Copyright [yyyy] [name of copyright owner]
19 #
20 # CDDL HEADER END
21 #
22 #
23 # Copyright (c) 1993 by Sun Microsystems, Inc.
24 #
25 # Copyright (c) 2018, Joyent, Inc.
26 # Copyright 2020 Peter Tribble.
27 #

28 PROG= setfacl

30 include ../Makefile.cmd

31 CERRWARN += _gcc=-Wno-unused-variable
32 CERRWARN += _gcc=-Wno-implicit-function-declaration
33 CERRWARN += $(CNOWARN_UNINIT)

35 # not linted
36 SMATCH=off

32 LDLIBS += -lsec

34 .KEEP_STATE:

36 all: $(PROG)

38 install: all $(ROOTPROG)

40 clean:

48 lint: lint_PROG

42 include ../Makefile.targ
```

```

*****
20408 Sat Feb 8 11:08:23 2020
new/usr/src/cmd/setfacl/setfacl.c
12288 getfacl and setfacl could stand improvement
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright (c) 1993, 2010, Oracle and/or its affiliates. All rights reserved.
23 * Copyright 2020 Peter Tribble.
24 */

26 /*
27 * setfacl [-r] -f aclfile file ...
28 * setfacl [-r] -d acl_entries file ...
29 * setfacl [-r] -m acl_entries file ...
30 * setfacl [-r] -s acl_entries file ...
31 * This command deletes/adds/modifies/sets discretionary information for a file
32 * or files.
33 */

35 #include <stdlib.h>
36 #include <stdio.h>
37 #include <pwd.h>
38 #include <grp.h>
39 #include <string.h>
40 #include <locale.h>
41 #include <sys/acl.h>
42 #include <sys/types.h>
43 #include <unistd.h>
44 #include <errno.h>
45 #include <ctype.h>

47 #define ADD 1
48 #define MODIFY 2
49 #define DELETE 3
50 #define SET 4

52 static int get_acl_info(char *filep, aclent_t **aclpp);
53 static int mod_entries(aclent_t *, int, char *, char *, char *, int);
54 static int set_file_entries(char *, char *, int);
55 static int set_online_entries(char *, char *, int);
56 static void usage();
57 static int parse_entry_list(aclent_t **, int *, char *, int);
58 static int convert_to_aclent_t(char *, int *, aclent_t **, int);
59 static int parse_entry(char *, aclent_t *, int);
60 static void err_handle(int, aclent_t *);
61 static int conv_id(char *);

```

```

63 int
64 main(int argc, char *argv[])
65 {
66     int c;
67     int dflag = 0;
68     int mflag = 0;
69     int rflag = 0;
70     int sflag = 0;
71     int fflag = 0;
72     int errflag = 0;
73     int aclcnt; /* used by -m -d */
74     aclent_t *aclp; /* used by -m -d */
75     char *aclfilep = NULL; /* acl file argument */
76     char *aclfile; /* acl file argument */
77     char *d_entryp = NULL; /* ptr to del entry list */
78     char *m_entryp = NULL; /* ptr to mod entry list */
79     char *s_entryp = NULL; /* ptr to set entry list */
80     char *work_dp = NULL; /* working ptrs for the above */
81     char *work_sp = NULL;

83     (void) setlocale(LC_ALL, "");
84     (void) textdomain(TEXT_DOMAIN);

86     if (argc < 3)
87         usage();

89     while ((c = getopt(argc, argv, "rm:d:s:f:")) != EOF) {
90         switch (c) {
91             case 'r':
92                 rflag++;
93                 break;
94             case 'd':
95                 if (dflag || fflag || sflag)
96                     usage();
97                 dflag++;
98                 d_entryp = optarg;
99                 break;
100             case 'm':
101                 if (mflag || fflag || sflag)
102                     usage();
103                 mflag++;
104                 m_entryp = optarg;
105                 break;
106             case 's':
107                 if (fflag || sflag || mflag || dflag)
108                     usage();
109                 sflag++;
110                 s_entryp = optarg;
111                 break;
112             case 'f':
113                 if (fflag || sflag || mflag || dflag)
114                     usage();
115                 fflag++;
116                 aclfilep = optarg;
117                 break;
118             case '?':
119                 errflag++;
120                 break;
121         }
122     }
123     if (errflag)
124         usage();

126     /* one of these flags should be set */

```

```

127     if (!fflag && !sflag && !mflag && !dflag)
128         usage();

130     /* no file arguments */
131     if (optind >= argc)
132         usage();

134     for (; optind < argc; optind++) {
135         register char *filep;

137         filep = argv[optind];

139         /* modify and delete: we need to get the ACL first */
140         if (mflag || dflag) {
141             if (m_entryp != NULL) {
142                 free(work_mp);
143                 work_mp = strdup(m_entryp);
144                 if (work_mp == NULL) {
145                     fprintf(stderr,
146                         gettext("out of memory %s\n"),
147                         m_entryp);
148                     exit(1);
149                 }
150             }

152             if (d_entryp != NULL) {
153                 free(work_dp);
154                 work_dp = strdup(d_entryp);
155                 if (work_dp == NULL) {
156                     fprintf(stderr,
157                         gettext("out of memory %s\n"),
158                         d_entryp);
159                     exit(1);
160                 }
161             }

163             aclcnt = get_acl_info(filep, &aclp);
164             if (aclcnt == -1)
165                 exit(2);
166             if (mod_entries(aclp, aclcnt, work_mp,
167                 work_dp, filep, rflag) == -1)
168                 exit(2);
169         } else if (fflag) {
170             if (set_file_entries(aclfilep, filep, rflag) == -1)
171                 exit(2);
172         } else if (sflag) {
173             if (s_entryp != NULL) {
174                 free(work_sp);
175                 work_sp = strdup(s_entryp);
176                 if (work_sp == NULL) {
177                     fprintf(stderr,
178                         gettext("out of memory %s\n"),
179                         s_entryp);
180                     exit(1);
181                 }
182             }
183             if (set_online_entries(work_sp, filep, rflag) == -1)
184                 exit(2);
185         }
186     }
187     return (0);
188 }

190 /*
191  * For add, modify, and delete, we need to get the ACL of the file first.
192  */

```

```

193 static int
194 get_acl_info(char *filep, aclent_t **aclpp)
195 {
196     int    aclcnt;

198     if ((aclcnt = acl(filep, GETACLNT, 0, NULL)) < 0) {
199         if (errno == ENOSYS) {
200             (void) fprintf(stderr,
201                 gettext("File system doesn't support aclent_t "
202                     "style ACL's.\n"
203                     "See acl(5) for more information on"
204                     "POSIX-draft ACL support.\n"));
205             (void) fprintf(stderr,
206                 gettext("  ACL styles support by Solaris.\n"));
207             return (-1);
208         }
209         (void) fprintf(stderr,
210             gettext("%s: failed to get acl count\n"), filep);
211         perror("get acl count error");
212         return (-1);
213     }
214     if (aclcnt < MIN_ACL_ENTRIES) {
215         (void) fprintf(stderr,
216             gettext("%d: acl count is too small from %s\n"),
217             aclcnt, filep);
218         return (-1);
219     }
220     if ((*aclpp = (aclent_t *)malloc(sizeof (aclent_t) * aclcnt)) == NULL) {
221         (void) fprintf(stderr, gettext("out of memory\n"));
222         return (-1);
223     }
224     if (acl(filep, GETACL, aclcnt, *aclpp) < 0) {
225         (void) fprintf(stderr,
226             gettext("%s: failed to get acl entries\n"), filep);
227         perror("getacl error");
228         return (-1);
229     }
230     return (aclcnt);
231 }

232 /*
233  * mod_entries() handles add, delete, and modify ACL entries of a file.
234  * The real action is in convert_to_aclent_t() called by parse_entry_list().
235  * aclp: points ACL of a file and may be changed by lower level routine.
236  * modp: modify entry list in ascii format
237  * delp: delete entry list in ascii format
238  * fnamep: file of interest
239  */
240 static int
241 mod_entries(aclent_t *aclp, int cnt, char *modp, char *delp,
242     char *fnamep, int rfg)
243 {
244     int    rc; /* return code */

245     /* modify and add: from -m option */
246     if (parse_entry_list(&aclp, &cnt, modp, MODIFY) == -1)
247         return (-1);

248     /* deletion: from -d option */
249     if (parse_entry_list(&aclp, &cnt, delp, DELETE) == -1)
250         return (-1);

251     if (aclsort(cnt, rfg, aclp) == -1) {
252         (void) err_handle(cnt, aclp);
253         (void) fprintf(stderr,
254             gettext("aclcnt %d, file %s\n"), cnt, fnamep);
255     }

```

```

256         return (-1);
257     }

259     if (acl(fnamep, SETACL, cnt, aclp) < 0) {
260         fprintf(stderr,
261             gettext("%s: failed to set acl entries\n"), fnamep);
262         perror("setacl error");
263         return (-1);
264     }
265     return (0);
266 }

```

unchanged portion omitted

```

336 /*
337 * set_online_entries() parses the acl entries from command line (setp).
338 * It converts the comma separated acl entries into aclent_t format.
339 * It then recalculates the mask according to rflag.
340 * Finally it sets ACL to the file (fnamep).
341 */
342 static int
343 set_online_entries(char *setp, char *fnamep, int rflag)
344 {
345     char          *commap;
346     aclent_t      *aclp;
347     int           aclcnt = 0;

348     if (parse_entry_list(&aclp, &aclcnt, setp, SET) == -1)
349         return (-1);

351     if (aclsort(aclcnt, rflag, aclp) == -1) {
352         (void) err_handle(aclcnt, aclp);
353         (void) fprintf(stderr,
354             gettext("aclcnt %d, file %s\n"), aclcnt, fnamep);
355         return (-1);
356     }

358     if (acl(fnamep, SETACL, aclcnt, aclp) < 0) {
359         fprintf(stderr,
360             gettext("%s: failed to set acl entries\n"), fnamep);
361         perror("setacl error");
362         return (-1);
363     }
364     return (0);
365 }

```

unchanged portion omitted

```

396 /*
397 * convert_to_aclent_t() converts an acl entry in ascii format (fields separated
398 * by colon) into aclent_t and appends it to the current ACL. It also handles
399 * memory allocation/deallocation for acl entries in aclent_t format.
400 * aclpp that contains acl entries in acl format will be returned.
401 * We don't check duplicates.
402 */
403 static int
404 convert_to_aclent_t(char *entryp, int *cntp, aclent_t **aclpp, int mode)
405 {
406     aclent_t      *new_aclp;
407     aclent_t      tmpacl;
408     aclent_t      *taclp, *centry = NULL, *gentry = NULL;
409     int           cur_cnt;
410     int           found = 0;
411     int           is_obj;

413     if (entryp == NULL)
414         return (0);

```

```

417     if (*cntp > 1)
418         new_aclp = (aclent_t *)realloc(*aclpp,
419             sizeof (aclent_t) * (*cntp));
420     else
421         new_aclp = (aclent_t *) malloc(sizeof (aclent_t) * (*cntp));
422     if (new_aclp == NULL) {
423         fprintf(stderr,
424             gettext("Insufficient memory for acl %d\n"), *cntp);
425         return (-1);
426     }

416     tmpacl.a_id = 0; /* id field needs to be initialized */
417     if (entryp[0] == 'u')
418         tmpacl.a_id = getuid(); /* id field for user */
419     if (entryp[0] == 'g')
420         tmpacl.a_id = getgid(); /* id field for group */

422     tmpacl.a_type = 0;
423     if (parse_entry(entryp, &tmpacl, mode) == -1)
424         return (-1);

426     is_obj = ((tmpacl.a_type == USER_OBJ) ||
427             (tmpacl.a_type == GROUP_OBJ) ||
428             (tmpacl.a_type == CLASS_OBJ) ||
429             (tmpacl.a_type == DEF_USER_OBJ) ||
430             (tmpacl.a_type == DEF_GROUP_OBJ) ||
431             (tmpacl.a_type == DEF_OTHER_OBJ));

433     if (*cntp > 1)
434         new_aclp = (aclent_t *)realloc(*aclpp,
435             sizeof (aclent_t) * (*cntp));
436     else
437         new_aclp = (aclent_t *) malloc(sizeof (aclent_t) * (*cntp));
438     if (new_aclp == NULL) {
439         fprintf(stderr,
440             gettext("Insufficient memory for acl %d\n"), *cntp);
441         return (-1);
442     }

444     cur_cnt = *cntp - 1;
445     switch (mode) {
446     case MODIFY: /* and add */
447         for (taclp = new_aclp; cur_cnt-- > 0; taclp++) {
448             if (taclp->a_type == tmpacl.a_type &&
449                 ((taclp->a_id == tmpacl.a_id) || is_obj)) {
450                 found++;
451                 /* cnt is added before it's called */
452                 *cntp -= 1;
453                 taclp->a_perm = tmpacl.a_perm;
454                 break;
455             }
456         }
457         if (!found) /* Add it to the end: no need to change cntp */
458             memcpy(new_aclp + *cntp - 1, &tmpacl, sizeof (aclent_t));
459         break;

461     case DELETE:
462         for (taclp = new_aclp; cur_cnt-- > 0; taclp++) {
463             if (taclp->a_type == tmpacl.a_type &&
464                 ((taclp->a_id == tmpacl.a_id) || is_obj)) {
465                 found++;
466                 /* move up the rest */
467                 while (cur_cnt-- > 0) {
468                     memcpy(taclp, taclp+1,
469                         sizeof (aclent_t));
470                     taclp++;

```

```

471     }
472     *cntp = *cntp - 2;
473     break;
474 }
475 }
476 if (!found)
477     *cntp -= 1;
478 break;

480 case SET:
481     /* we may check duplicate before copying over?? */
482     memcpy(new_aclp + *cntp - 1, &tmpacl, sizeof (aclent_t));
483     break;

485 default:
486     fprintf(stderr,
487             gettext("Unrecognized mode: internal error\n"));
488     break;
489 }

491 /*
492  * If converting from non-trivial acl entry to trivial one,
493  * reset CLASS_OBJ's permission with that of GROUP_OBJ.
494  */

496 if (mode == DELETE) {
497     boolean_t    trivial = B_TRUE;    /* assumption */
498     cur_cnt = *cntp;
499     for (taclp = new_aclp; cur_cnt-- > 0; taclp++) {
500         switch (taclp->a_type) {
501             case USER_OBJ:
502             case OTHER_OBJ:
503                 break;
504             case CLASS_OBJ:
505                 centry = taclp;
506                 break;
507             case GROUP_OBJ:
508                 gentry = taclp;
509                 break;
510             default:
511                 /*
512                  * Confirmed that the new acl set is
513                  * still a non-trivial acl.
514                  * Skip reset.
515                  */
516                 trivial = B_FALSE;
517         }
518     }
519     if (centry != NULL && gentry != NULL && trivial == B_TRUE)
520         centry->a_perm = gentry->a_perm;
521 }
522 *aclpp = new_aclp;    /* return new acl entries */
523 return (0);
524 }

```

unchanged portion omitted

```

*****
7842 Sat Feb 8 11:08:23 2020
new/usr/src/man/man1/getfacl.1
12288 getfacl and setfacl could stand improvement
*****
1 \" te
2 .\" \&.Copyright (c) 2002, Sun Microsystems, Inc. All Rights Reserved
3 .\" Copyright (c) 2020 Peter Tribble.
4 .\" The contents of this file are subject to the terms of the Common Development
5 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
6 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
7 .TH GETFACL 1 "Feb 8, 2020"
8 .TH GETFACL 1 "Nov 5, 1994"
9 .SH NAME
10 getfacl \- display discretionary file information
11 .SH SYNOPSIS
12 .LP
13 .nf
14 \fBgetfacl\fR [\fB-ad\fR] \fIfile\fR...
15 .fi
16 .SH DESCRIPTION
17 .sp
18 .LP
19 For each argument that is a regular file, special file, or named pipe, the
20 \fBgetfacl\fR utility displays the owner, the group, and the Access Control
21 List (\fBACL\fR). For each directory argument, \fBgetfacl\fR displays the
22 owner, the group, and the \fBACL\fR and/or the default \fBACL\fR. Only
23 directories contain default \fBACL\fRs.
24 .sp
25 .LP
26 The \fBgetfacl\fR utility will fail if executed on a file system that supports
27 NFSv4 \fBACL\fRs. See \fBacl\fR(5) for a description of the difference
28 between the older POSIX-draft \fBACL\fRs and the newer NFSv4 \fBACL\fRs. The
29 \fBls\fR(1) utility, when used with the \fB-v\fR or \fB-V\fR options, will
30 display \fBACL\fRs on all types of file system.
31 .sp
32 .LP
33 The \fBgetfacl\fR utility may be executed on a file system that does not
34 support \fBACL\fRs. It reports the \fBACL\fR based on the base permission bits.
35 .sp
36 .LP
37 With no options specified, \fBgetfacl\fR displays the filename, the file owner,
38 the file group owner, and both the \fBACL\fR and the default \fBACL\fR, if it
39 exists.
40 .SH OPTIONS
41 .sp
42 .LP
43 The following options are supported:
44 .sp
45 .ne 2
46 .na
47 \fB-a\fR
48 .ad
49 .RS 6n
50 Displays the filename, the file owner, the file group owner, and the \fBACL\fR
51 of the file.
52 .RE
53 .sp
54 .ne 2
55 .na
56 \fB-d\fR
57 .ad
58 .RS 6n
59 Displays the filename, the file owner, the file group owner, and the default

```

```

56 \fBACL\fR of the file, if it exists.
57 .RE
58 .SH OPERANDS
59 .sp
60 .LP
61 The following operands are supported:
62 .sp
63 .na
64 \fB-f\fR \fIfile\fR
65 .ad
66 .RS 8n
67 The path name of a regular file, special file, or named pipe.
68 .RE
69 .SH OUTPUT
70 .sp
71 .LP
72 The format for \fBACL\fR output is as follows:
73 .sp
74 .in +2
75 .nf
76 # file: filename
77 # owner: uid
78 # group: gid
79 user::perm
80 user:uid:perm
81 group:perm
82 group:gid:perm
83 mask:perm
84 other:perm
85 default:user::perm
86 default:user:uid:perm
87 default:group:perm
88 default:group:gid:perm
89 default:mask:perm
90 default:other:perm
91 .fi
92 .in -2
93 .sp
94 .sp
95 .LP
96 When multiple files are specified on the command line, a blank line separates
97 the \fBACL\fRs for each file.
98 .sp
99 .LP
100 The \fBACL\fR entries are displayed in the order in which they are evaluated
101 when an access check is performed. The default \fBACL\fR entries that may exist
102 on a directory have no effect on access checks.
103 .sp
104 .LP
105 The first three lines display the filename, the file owner, and the file group
106 owner. Notice that when only the \fB-d\fR option is specified and the file has
107 no default \fBACL\fR, only these three lines are displayed.
108 .sp
109 .LP
110 The \fBuser\fR entry without a user \fBID\fR indicates the permissions that
111 are granted to the file owner. One or more additional user entries indicate the
112 permissions that are granted to the specified users.
113 .sp
114 .LP
115 The \fBgroup\fR entry without a group \fBID\fR indicates the permissions that
116 are granted to the file group owner. One or more additional group entries
117 indicate the permissions that are granted to the specified groups.

```

```

118 .sp
119 .LP
120 The \fBmask\fR entry indicates the \fBACL\fR mask permissions. These are the
121 maximum permissions allowed to any user entries except the file owner, and to
122 any group entries, including the file group owner. These permissions restrict
123 the permissions specified in other entries.
124 .sp
125 .LP
126 The \fBOther\fR entry indicates the permissions that are granted to others.
127 .sp
128 .LP
129 The \fBdefault\fR entries may exist only for directories. These entries
130 indicate the default entries that are added to a file created within the
131 directory.
132 .sp
133 .LP
134 The \fBuid\fR is a login name or a user \fBID\fR if there is no entry for the
135 \fBuid\fR in the system password file, \fB/etc/passwd\fR. The \fBgid\fR is a
136 group name or a group \fBID\fR if there is no entry for the \fBgid\fR in the
137 system group file, \fB/etc/group\fR. The \fBperm\fR is a three character string
138 composed of the letters representing the separate discretionary access rights:
139 \fBr\fR (read), \fBw\fR (write), \fBx\fR (execute/search), or the place holder
140 character \fB(mi)\fR. The \fBperm\fR is displayed in the following order:
141 \fBrwx\fR. If a permission is not granted by an \fBACL\fR entry, the place
142 holder character appears.
143 .sp
144 .LP
145 If you use the \fBchmod\fR(1) command to change the file group owner
146 permissions on a file with \fBACL\fR entries, both the file group owner
147 permissions and the \fBACL\fR mask are changed to the new permissions. Be aware
148 that the new \fBACL\fR mask permissions may change the effective permissions
149 for additional users and groups who have \fBACL\fR entries on the file.
150 .sp
151 .LP
152 In order to indicate that the \fBACL\fR mask restricts an \fBACL\fR entry,
153 \fBgetfacl\fR displays an additional tab character, pound sign (\fB#\fR), and
154 the actual permissions granted, following the entry.
155 .SH EXAMPLES
156 .LP
157 \fBExample 1 \fRDisplaying file information
158 .sp
159 Given file \fBfoo\fR, with an \fBACL\fR six entries long, the command

```

```

161 .sp
162 .in +2
163 .nf
164 host% \fBgetfacl foo\fR
165 .fi
166 .in -2
167 .sp

169 .sp
170 .LP
171 would print:

173 .sp
174 .in +2
175 .nf
176 # file: foo
177 # owner: shea
178 # group: staff
179 user::rwx
180 user:spy:\|\(mi)\|\(mi)\|\(mi)
181 user:mookie:r\|\(mi)\|\(mi)
182 group:r\|\(mi)\|\(mi)

```

```

183 mask::rw\|\(mi
184 other::\|\(mi)\|\(mi)\|\(mi
185 .fi
186 .in -2
187 .sp

189 .LP
190 \fBExample 2 \fRDisplaying information after chmod command
191 .sp
192 .LP
193 Continue with the above example, after \fBchmod\fR \fB700 foo\fR was issued:

195 .sp
196 .in +2
197 .nf
198 host% \fBgetfacl foo\fR
199 .fi
200 .in -2
201 .sp

203 .sp
204 .LP
205 would print:

207 .sp
208 .in +2
209 .nf
210 # file: foo
211 # owner: shea
212 # group: staff
213 user::rwx
214 user:spy:\|\(mi)\|\(mi)\|\(mi)
215 user:mookie:r\|\(mi)\|\(mi) #effective:\|\(mi)\|\(mi)\|\(mi
216 group::\|\(mi)\|\(mi)\|\(mi
217 mask::\|\(mi)\|\(mi)\|\(mi
218 other::\|\(mi)\|\(mi)\|\(mi
219 .fi
220 .in -2
221 .sp

223 .LP
224 \fBExample 3 \fRDisplaying information when ACL contains default entries
225 .sp
226 .LP
227 Given directory \fBdoo\fR, with an \fBACL\fR containing default entries, the
228 command

230 .sp
231 .in +2
232 .nf
233 host% \fBgetfacl -d doo\fR
234 .fi
235 .in -2
236 .sp

238 .sp
239 .LP
240 would print:

242 .sp
243 .in +2
244 .nf
245 # file: doo
246 # owner: shea
247 # group: staff
248 default:user::rwx

```

```

249 default:user:spy:\|\(mi\|\(mi\|\(mi
250 default:user:mookie:r\|\(mi\|\(mi
251 default:group:r\|\(mi\|\(mi
252 default:mask:\|\(mi\|\(mi\|\(mi
253 default:other::\|\(mi\|\(mi\|\(mi
254 .fi
255 .in -2
256 .sp

258 .SH FILES
261 .sp
259 .ne 2
260 .na
261 \fB\fB/etc/passwd\fR\fR
262 .ad
263 .RS 15n
264 system password file
265 .RE

267 .sp
268 .ne 2
269 .na
270 \fB\fB/etc/group\fR\fR
271 .ad
272 .RS 15n
273 group file
274 .RE

276 .SH ATTRIBUTES
280 .sp
281 .LP
277 See \fBattributes\fR(5) for descriptions of the following attributes:
278 .sp

280 .sp
281 .TS
282 box;
283 c | c
284 l | l .
285 ATTRIBUTE TYPE ATTRIBUTE VALUE
286 _
287 Interface Stability Evolving
288 .TE

290 .SH SEE ALSO
296 .sp
297 .LP
291 \fBbchmod\fR(1), \fBbfs\fR(1), \fBbsetfacl\fR(1), \fBbacl\fR(2),
292 \fBbaclsort\fR(3SEC), \fBbgroup\fR(4), \fBbpasswd\fR(4), \fBbacl\fR(5),
293 \fBattributes\fR(5)
299 \fBbaclsort\fR(3SEC), \fBbgroup\fR(4), \fBbpasswd\fR(4), \fBattributes\fR(5)
294 .SH NOTES
301 .sp
302 .LP
295 The output from \fBgetfacl\fR is in the correct format for input to the
296 \fBsetfacl\fR \fB-f\fR command. If the output from \fBgetfacl\fR is redirected
297 to a file, the file may be used as input to \fBsetfacl\fR. In this way, a user
298 may easily assign one file's \fBACL\fR to another file.

```

```

*****
11823 Sat Feb 8 11:08:23 2020
new/usr/src/man/man1/setfacl.1
12288 getfacl and setfacl could stand improvement
*****
1 \" te
2.\" Copyright (c) 2006, Sun Microsystems, Inc. All Rights Reserved
3.\" Copyright (c) 2020 Peter Tribble.
4.\" The contents of this file are subject to the terms of the Common Development
5.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
6.\" When distributing Covered Code, include this CDDL HEADER in each file and in
7.TH SETFACL 1 "Feb 8, 2020"
6.TH SETFACL 1 "Dec 19, 2006"
8.SH NAME
9 setfacl \- modify the Access Control List (ACL) for a file or files
10.SH SYNOPSIS
10.LP
11.nf
12 \fbsetfacl\fr [[\fb-r\fr] \fb-s\fr \fIacl_entries\fr \fIfile\fr
13 .fi

15.LP
16.nf
17 \fbsetfacl\fr [[\fb-r\fr] \fb-md\fr \fIacl_entries\fr \fIfile\fr
18 .fi

20.LP
21.nf
22 \fbsetfacl\fr [[\fb-r\fr] \fb-f\fr \fIacl_file\fr \fIfile\fr
23 .fi

25.SH DESCRIPTION
26.sp
27.LP
26 For each file specified, \fbsetfacl\fr either replaces its entire \fBACL\fr,
27 including the default \fBACL\fr on a directory, or it adds, modifies, or
28 deletes one or more \fBACL\fr entries, including default entries on
29 directories.
30.sp
31.LP
32 The \fbsetfacl\fr utility can only manipulate POSIX-draft \fBACL\frs. See
33 \fBacl\fr(5) for a description of the difference between the older POSIX-draft
34 \fBACL\frs and the newer NFSv4 \fBACL\frs. The \fbchmod\fr(1) utility can
35 be used to manipulate \fBACL\frs on all types of file system.
36.sp
37.LP
38 When the \fbsetfacl\fr command is used, it can result in changes to the file
39 permission bits. When the user \fBACL\fr entry for the file owner is changed,
40 the file owner class permission bits are modified. When the group \fBACL\fr
41 entry for the file group class is changed, the file group class permission bits
42 are modified. When the other \fBACL\fr entry is changed, the file other class
43 permission bits are modified.
44.sp
45.LP
46 If you use the \fbchmod\fr(1) command to change the file group owner
47 permissions on a file with \fBACL\fr entries, both the file group owner
48 permissions and the \fBACL\fr mask are changed to the new permissions. Be aware
49 that the new \fBACL\fr mask permissions can change the effective permissions
50 for additional users and groups who have \fBACL\fr entries on the file.
51.sp
52.LP
53 A directory can contain default \fBACL\fr entries. If a file or directory is
54 created in a directory that contains default \fBACL\fr entries, the newly
55 created file has permissions generated according to the intersection of the
56 default \fBACL\fr entries and the permissions requested at creation time. The
57 \fbumask\fr(1) are not applied if the directory contains default \fBACL\fr

```

```

58 entries. If a default \fBACL\fr is specified for a specific user (or users),
59 the file has a regular \fBACL\fr created. Otherwise, only the mode bits are
60 initialized according to the intersection described above. The default
61 \fBACL\fr should be thought of as the maximum discretionary access permissions
62 that can be granted.
63.sp
64.LP
65 Use the \fbsetfacl\fr command to set ACLs on files in a UFS file system, which
66 supports POSIX-draft ACLS (or \fbaclet_t\fr style ACLs). Use the \fbchmod\fr
67 command to set ACLs on files in a ZFS file system, which supports NFSv4-style
68 ACLS (or \fbace_t\fr style ACLs).
69.SS "\fIacl_entries\fr Syntax"
66.sp
67.LP
70 For the \fb-m\fr and \fb-s\fr options, \fIacl_entries\fr are one or more
71 comma-separated \fBACL\fr entries.
72.sp
73.LP
74 An \fBACL\fr entry consists of the following fields separated by colons:
75.sp
76.ne 2
77.na
78 \fb\fientry_type\fr\fr
79.ad
80.RS 14n
81 Type of \fBACL\fr entry on which to set file permissions. For example,
82 \fientry_type\fr can be \fbuser\fr (the owner of a file) or \fbmask\fr (the
83 \fBACL\fr mask).
84.RE

86.sp
87.ne 2
88.na
89 \fb\fiuid\fr or \fIgid\fr\fr
90.ad
91.RS 14n
92 User name or user identification number. Or, group name or group identification
93 number.
94.RE

96.sp
97.ne 2
98.na
99 \fb\fiperms\fr\fr
100.ad
101.RS 14n
102 Represents the permissions that are set on \fientry_type\fr. \fiperms\fr can be
103 indicated by the symbolic characters \fbRwx\fr or a number (the same
104 permissions numbers used with the \fbchmod\fr command).
105.RE

107.sp
108.LP
109 The following table shows the valid \fBACL\fr entries (default entries can only
110 be specified for directories):
111.sp

113.sp
114.TS
115.c c
116.l l .
117 \fBACL\fr Entry Description
118 _
119 u[ser]::\fiperms\fr File owner permissions.
120 g[roup]::\fiperms\fr File group owner permissions.
121 o[ther]::\fiperms\fr T{

```

```

122 Permissions for users other than the file owner or members of file group owner.
123 T}
124 m[ask]:\fIperms\fR T{
125 The \fBACL\fR mask. The mask entry indicates the maximum permissions allowed for
126 T}
127 u[ser]:\fIuid:perms\fR T{
128 Permissions for a specific user. For \fIuid\fR, you can specify either a user na
129 T}
130 g[roup]:\fIgid:perms\fR T{
131 Permissions for a specific group. For \fIgid\fR, you can specify either a group
132 T}
133 d[efault]:u[ser]:\fIperms\fR Default file owner permissions.
134 d[efault]:g[roup]:\fIperms\fR Default file group owner permissions.
135 d[efault]:o[ther]:\fIperms\fR T{
136 Default permissions for users other than the file owner or members of the file g
137 T}
138 d[efault]:m[ask]:\fIperms\fR Default \fBACL\fR mask.
139 d[efault]:u[ser]:\fIuid\fR:\fIperms\fR T{
140 Default permissions for a specific user. For \fIuid\fR, you can specify either a
141 T}
142 d[efault]:g[roup]:\fIgid\fR:\fIperms\fR T{
143 Default permissions for a specific group. For \fIgid\fR, you can specify either
144 T}
145 .TE

```

```

147 .sp
148 .LP
149 For the \fB-d\fR option, \fIacl_entries\fR are one or more comma-separated
150 \fBACL\fR entries without permissions. Notice that the entries for file owner,
151 file group owner, \fBACL\fR mask, and others can not be deleted.

```

#### 152 .SH OPTIONS

```

151 .sp
152 .LP
153 The options have the following meaning:

```

```

154 .sp
155 .ne 2
156 .na
157 \fB-fB-d\fR \fIacl_entries\fR\fR
158 .ad
159 .RS 18n
160 Deletes one or more entries from the file. The entries for the file owner, the
161 file group owner, and others can not be deleted from the \fBACL\fR. Notice that
162 deleting an entry does not necessarily have the same effect as removing all
163 permissions from the entry.
164 .RE

```

```

166 .sp
167 .ne 2
168 .na
169 \fB-fB-f\fR \fIacl_file\fR\fR
170 .ad
171 .RS 18n
172 Sets a file's \fBACL\fR with the \fBACL\fR entries contained in the file named
173 \fIacl_file\fR. The same constraints on specified entries hold as with the
174 \fB-s\fR option. The entries are not required to be in any specific order in
175 the file. Also, if you specify a dash (\fB-\fR) for \fIacl_file\fR, standard
176 input is used to set the file's \fBACL\fR.
177 .sp
178 The character \fB#\fR in \fIacl_file\fR can be used to indicate a comment. All
179 characters, starting with the \fB#\fR until the end of the line, are ignored.
180 Notice that if the \fIacl_file\fR has been created as the output of the
181 \fBgetfacl\fR(1) command, any effective permissions, which follow a \fB#\fR,
182 are ignored.
183 .RE

```

```

185 .sp

```

```

186 .ne 2
187 .na
188 \fB-fB-m\fR \fIacl_entries\fR\fR
189 .ad
190 .RS 18n
191 Adds one or more new \fBACL\fR entries to the file, and/or modifies one or more
192 existing \fBACL\fR entries on the file. If an entry already exists for a
193 specified \fIuid\fR or \fIgid\fR, the specified permissions replace the current
194 permissions. If an entry does not exist for the specified \fIuid\fR or
195 \fIgid\fR, an entry is created. When using the \fB-m\fR option to modify a
196 default \fBACL\fR, you must specify a complete default \fBACL\fR (user, group,
197 other, mask, and any additional entries) the first time.
198 .RE

```

```

200 .sp
201 .ne 2
202 .na
203 \fB-fB-r\fR\fR
204 .ad
205 .RS 18n
206 Recalculates the permissions for the \fBACL\fR mask entry. The permissions
207 specified in the \fBACL\fR mask entry are ignored and replaced by the maximum
208 permissions necessary to grant the access to all additional user, file group
209 owner, and additional group entries in the \fBACL\fR. The permissions in the
210 additional user, file group owner, and additional group entries are left
211 unchanged.
212 .RE

```

```

214 .sp
215 .ne 2
216 .na
217 \fB-fB-s\fR \fIacl_entries\fR\fR
218 .ad
219 .RS 18n
220 Sets a file's \fBACL\fR. All old \fBACL\fR entries are removed and replaced
221 with the newly specified \fBACL\fR. The entries need not be in any specific
222 order. They are sorted by the command before being applied to the file.

```

#### 223 .sp

```

224 Required entries:

```

```

225 .RS +4
226 .TP
227 .ie t \(\bu
228 .el o
229 Exactly one \fBuser\fR entry specified for the file owner.
230 .RE
231 .RS +4
232 .TP

```

```

233 .ie t \(\bu
234 .el o
235 Exactly one \fBgroup\fR entry for the file group owner.
236 .RE

```

```

237 .RS +4

```

```

238 .TP
239 .ie t \(\bu
240 .el o
241 Exactly one \fBother\fR entry specified.
242 .RE

```

```

243 If there are additional user and group entries:

```

```

244 .RS +4
245 .TP
246 .ie t \(\bu
247 .el o
248 Exactly one \fBmask\fR entry specified for the \fBACL\fR mask that indicates
249 the maximum permissions allowed for users (other than the owner) and groups.
250 .RE

```

```

251 .RS +4

```

```

252 .TP
253 .ie t \(\bu
254 .el o
255 Must not be duplicate \fBuser\fR entries with the same \fIuid\fR.
256 .RE
257 .RS +4
258 .TP
259 .ie t \(\bu
260 .el o
261 Must not be duplicate \fBgroup\fR entries with the same \fIgid\fR.
262 .RE
263 If \fIfile\fR is a directory, the following default \fBACL\fR entries can be
264 specified:
265 .RS +4
266 .TP
267 .ie t \(\bu
268 .el o
269 Exactly one \fBdefault user\fR entry for the file owner.
270 .RE
271 .RS +4
272 .TP
273 .ie t \(\bu
274 .el o
275 Exactly one \fBdefault group\fR entry for the file group owner.
276 .RE
277 .RS +4
278 .TP
279 .ie t \(\bu
280 .el o
281 Exactly one \fBdefault mask\fR entry for the \fBACL\fR mask.
282 .RE
283 .RS +4
284 .TP
285 .ie t \(\bu
286 .el o
287 Exactly one \fBdefault other\fR entry.
288 .RE
289 There can be additional \fBdefault user\fR entries and additional \fBdefault
290 group\fR entries specified, but there can not be duplicate additional
291 \fBdefault user\fR entries with the same \fIuid\fR, or duplicate \fBdefault
292 group\fR entries with the same \fIgid\fR.
293 .RE

295 .SH EXAMPLES
296 .LP
296 \fBExample 1\fR Adding read permission only
297 .sp
298 .LP
299 The following example adds one \fBACL\fR entry to file \fBabc\fR, which gives
300 user \fBshea\fR read permission only.

302 .sp
303 .in +2
304 .nf
305 \fBsetfacl -m user:shea:r\(\mi\(\mi abc\fR
306 .fi
307 .in -2
308 .sp

310 .LP
311 \fBExample 2\fR Replacing a file's entire \fBACL\fR
312 .sp
313 .LP
314 The following example replaces the entire \fBACL\fR for the file \fBabc\fR,
315 which gives \fBshea\fR read access, the file owner all access, the file group
316 owner read access only, the \fBACL\fR mask read access only, and others no

```

```

317 access.

319 .sp
320 .in +2
321 .nf
322 \fBsetfacl -s user:shea:rw, user::rw, group::rw-, mask:r--, other:--- abc\fR
323 .fi
324 .in -2
325 .sp

327 .sp
328 .LP
329 Notice that after this command, the file permission bits are \fBbrwxr----\fR.
330 Even though the file group owner was set with read/write permissions, the
331 \fBACL\fR mask entry limits it to have only read permission. The mask entry
332 also specifies the maximum permissions available to all additional user and
333 group \fBACL\fR entries. Once again, even though the user \fBshea\fR was set
334 with all access, the mask limits it to have only read permission. The \fBACL\fR
335 mask entry is a quick way to limit or open access to all the user and group
336 entries in an \fBACL\fR. For example, by changing the mask entry to read/write,
337 both the file group owner and user \fBshea\fR would be given read/write access.

339 .LP
340 \fBExample 3\fR Setting the same \fBACL\fR on two files
341 .sp
342 .LP
343 The following example sets the same \fBACL\fR on file \fBabc\fR as the file
344 \fBxyz\fR.

346 .sp
347 .in +2
348 .nf
349 \fBgetfacl xyz | setfacl -f \(\mi abc\fR
350 .fi
351 .in -2
352 .sp

354 .SH FILES
355 .sp
355 .ne 2
356 .na
357 \fB/etc/passwd\fR
358 .ad
359 .RS 15n
360 password file
361 .RE

363 .sp
364 .ne 2
365 .na
366 \fB/etc/group\fR
367 .ad
368 .RS 15n
369 group file
370 .RE

372 .SH SEE ALSO
373 .sp
374 .LP
373 \fBchmod\fR(1), \fBgetfacl\fR(1), \fBumask\fR(1), \fBaclcheck\fR(3SEC),
374 \fBaclsort\fR(3SEC), \fBgroup\fR(4), \fBpasswd\fR(4), \fBacl\fR(5),
375 \fBattributes\fR(5)
378 \fBaclsort\fR(3SEC), \fBgroup\fR(4), \fBpasswd\fR(4), \fBattributes\fR(5)

```

```

*****
11921 Sat Feb  8 11:08:23 2020
new/usr/src/man/man3sec/acl_totext.3sec
12288 getfacl and setfacl could stand improvement
*****
1 \" te
2 .\" Copyright (c) 20068 Sun Microsystems, Inc. All Rights Reserved.
3 .\" The contents of this file are subject to the terms of the Common Development
4 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
5 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
6 .TH ACL_TOTEXT 3SEC \"Jun 16, 2008\"
7 .SH NAME
8 acl_totext, acl_fromtext \- convert internal representation to or from
9 external representation
10 .SH SYNOPSIS
11 .LP
12 cc [ \fiflag\fR&.\|. \| . ] \fifile\fR&.\|. \| . \fB-lsec\fR [ \fIlibrary\fR&.\|. \| .
13 #include <sys/acl.h>

15 \fBchar *\fR\fBacl_totext\fR(\fBacl_t *\fR\fIaclp\fR, \fBint\fR \fIflags\fR);
16 .fi

18 .LP
19 .nf
20 \fBint\fR \fBacl_fromtext\fR(\fBchar *\fR\fIacltextp\fR, \fBacl_t **\fR\fIaclp\fR
21 .fi

23 .SH DESCRIPTION
25 .sp
26 .LP
24 The \fBacl_totext()\fR function converts an internal ACL representation pointed
25 to by \fIaclp\fR into an external ACL representation. The memory for the
26 external text string is obtained using \fBmalloc\fR(3C). The caller is
27 responsible for freeing the memory upon completion.
28 .sp
29 .LP
30 The format of the external ACL is controlled by the \fIflags\fR argument.
31 Values for \fIflags\fR are constructed by a bitwise-inclusive-OR of \fIflags\fR
32 from the following list, defined in <\fBsys/acl.h\fR>.
33 .sp
34 .ne 2
35 .na
36 \fB\fBACL_COMPACT_FMT\fR\fR
37 .ad
38 .RS 19n
39 For NFSv4 ACLs, the ACL entries will be formatted using the compact ACL format
40 detailed in \fBls\fR(1) for the \fB-V\fR option.
41 .RE

43 .sp
44 .ne 2
45 .na
46 \fB\fBACL_APPEND_ID\fR\fR
47 .ad
48 .RS 19n
49 Append the \fBuid\fR or \fBgid\fR for additional user or group entries. This
50 flag is used to construct ACL entries in a manner that is suitable for archive
51 utilities such as \fBtar\fR(1). When the ACL is translated from the external
52 format to internal representation using \fBacl_fromtext()\fR, the appended ID
53 will be used to populate the \fBuid\fR or \fBgid\fR field of the ACL entry when
54 the user or group name does not exist on the host system. The appended id will
55 be ignored when the user or group name does exist on the system.
56 .RE

```

```

58 .sp
59 .ne 2
60 .na
61 \fB\fBACL_SID_FMT\fR\fR
62 .ad
63 .RS 19n
64 For NFSv4 ACLs, the ACL entries for user or group entries will use the
65 \fBusersid\fR or \fBgroupsid\fR format when the "id" field in the ACL entry is
66 an ephemeral \fBuid\fR or \fBgid\fR. The raw \fBsid\fR format will only be
67 used when the "id" cannot be resolved to a windows name.
68 .RE

70 .sp
71 .LP
72 The \fBacl_fromtext()\fR function converts an external ACL representation
73 pointed to by \fIacltextp\fR into an internal ACL representation. The memory
74 for the list of ACL entries is obtained using \fBmalloc\fR(3C). The caller is
75 responsible for freeing the memory upon completion. Depending on type of ACLs a
76 file system supports, one of two external external representations are
77 possible. For POSIX draft file systems such as ufs, the external representation
78 is described in \fBacltotext\fR(3SEC). The external ACL representation For
79 NFSv4-style ACLs is detailed as follows.
80 .sp
81 .LP
82 Each \fBacl_entry\fR contains one ACL entry. The external representation of an
83 ACL entry contains three, four or five colon separated fields. The first field
84 contains the ACL entry type. The entry type keywords are defined as:
85 .sp
86 .ne 2
87 .na
88 \fB\fBEveryone@\fR\fR
89 .ad
90 .RS 13n
91 This ACL entry specifies the access granted to any user or group that does not
92 match any previous ACL entry.
93 .RE

95 .sp
96 .ne 2
97 .na
98 \fB\fBGroup\fR\fR
99 .ad
100 .RS 13n
101 This ACL entry with a GID specifies the access granted to a additional group of
102 the object.
103 .RE

105 .sp
106 .ne 2
107 .na
108 \fB\fBGroup@\fR\fR
109 .ad
110 .RS 13n
111 This ACL entry with no GID specified in the ACL entry field specifies the
112 access granted to the owning group of the object.
113 .RE

115 .sp
116 .ne 2
117 .na
118 \fB\fBGroupsid\fR\fR
119 .ad
120 .RS 13n
121 This ACL entry with a SID or Windows name specifies the access granted to a
122 Windows group. This type of entry is for a CIFS server created file.
123 .RE

```

```

125 .sp
126 .ne 2
127 .na
128 \fB\fBowner@\fR\fR
129 .ad
130 .RS 13n
131 This ACL entry with no UID specified in the ACL entry field specifies the
132 access granted to the owner of the object.
133 .RE

135 .sp
136 .ne 2
137 .na
138 \fB\fBsid\fR\fR
139 .ad
140 .RS 13n
141 This ACL entry with a SID or Windows name when the entry could be either a
142 group or a user.
143 .RE

145 .sp
146 .ne 2
147 .na
148 \fB\fBuser@\fR\fR
149 .ad
150 .RS 13n
151 This ACL entry with a UID specifies the access granted to a additional user of
152 the object.
153 .RE

155 .sp
156 .ne 2
157 .na
158 \fB\fBusersid\fR\fR
159 .ad
160 .RS 13n
161 This ACL entry with a SID or Windows name specifies the access granted to a
162 Windows user. This type of entry is for a CIFS server created file.
163 .RE

165 .sp
166 .LP
167 The second field contains the ACL entry ID, and is used only for user or group
168 ACL entries. This field is not used for \fBowner@\fR, \fBgroup@\fR, or
169 \fBeveryone@\fR entries.
170 .sp
171 .ne 2
172 .na
173 \fB\fBuid\fR\fR
174 .ad
175 .RS 7n
176 This field contains a user-name or user-ID. If the user-name cannot be resolved
177 to a UID, then the entry is assumed to be a numeric UID.
178 .RE

180 .sp
181 .ne 2
182 .na
183 \fB\fBgid\fR\fR
184 .ad
185 .RS 7n
186 This field contains a group-name or group-ID. If the group-name can't be
187 resolved to a GID, then the entry is assumed to be a numeric GID.
188 .RE

```

```

190 .sp
191 .LP
192 The third field contains the discretionary access permissions. The format of
193 the permissions depends on whether \fBACL_COMPACT_FMT\fR is specified. When the
194 \fBiflags\fR field does not request \fBACL_COMPACT_FMT\fR, the following format
195 is used with a forward slash (/) separating the permissions.
196 .sp
197 .ne 2
198 .na
199 \fB\fBadd_file\fR\fR
200 .ad
201 .RS 20n
202 Add a file to a directory.
203 .RE

205 .sp
206 .ne 2
207 .na
208 \fB\fBadd_subdirectory\fR\fR
209 .ad
210 .RS 20n
211 Add a subdirectory.
212 .RE

214 .sp
215 .ne 2
216 .na
217 \fB\fBappend\fR\fR
218 .ad
219 .RS 20n
220 Append data.
221 .RE

223 .sp
224 .ne 2
225 .na
226 \fB\fBdelete\fR\fR
227 .ad
228 .RS 20n
229 Delete.
230 .RE

232 .sp
233 .ne 2
234 .na
235 \fB\fBdelete_child\fR\fR
236 .ad
237 .RS 20n
238 Delete child.
239 .RE

241 .sp
242 .ne 2
243 .na
244 \fB\fBexecute\fR\fR
245 .ad
246 .RS 20n
247 Execute permission.
248 .RE

250 .sp
251 .ne 2
252 .na
253 \fB\fBlist_directory\fR\fR
254 .ad
255 .RS 20n

```

```

256 List a directory.
257 .RE

259 .sp
260 .ne 2
261 .na
262 \fB\fBread_acl\fR\fR
263 .ad
264 .RS 20n
265 Read ACL.
266 .RE

268 .sp
269 .ne 2
270 .na
271 \fB\fBread_data\fR\fR
272 .ad
273 .RS 20n
274 Read permission.
275 .RE

277 .sp
278 .ne 2
279 .na
280 \fB\fBread_attributes\fR\fR
281 .ad
282 .RS 20n
283 Read attributes.
284 .RE

286 .sp
287 .ne 2
288 .na
289 \fB\fBread_xattr\fR\fR
290 .ad
291 .RS 20n
292 Read named attributes.
293 .RE

295 .sp
296 .ne 2
297 .na
298 \fB\fBrsynchronize\fR\fR
299 .ad
300 .RS 20n
301 Synchronize.
302 .RE

304 .sp
305 .ne 2
306 .na
307 \fB\fBwrite_acl\fR\fR
308 .ad
309 .RS 20n
310 Write ACL.
311 .RE

313 .sp
314 .ne 2
315 .na
316 \fB\fBwrite_attributes\fR\fR
317 .ad
318 .RS 20n
319 Write attributes.
320 .RE

```

```

322 .sp
323 .ne 2
324 .na
325 \fB\fBwrite_data\fR\fR
326 .ad
327 .RS 20n
328 Write permission.
329 .RE

331 .sp
332 .ne 2
333 .na
334 \fB\fBwrite_owner\fR\fR
335 .ad
336 .RS 20n
337 Write owner.
338 .RE

340 .sp
341 .ne 2
342 .na
343 \fB\fBwrite_xattr\fR\fR
344 .ad
345 .RS 20n
346 Write named attributes.
347 .RE

349 .sp
350 .LP
351 This format allows permissions to be specified as, for example:
352 \fBread_data\fR/\fBread_xattr\fR/\fBread_attributes\fR.
353 .sp
354 .LP
355 When \fBACL_COMPACT_FMT\fR is specified, the permissions consist of 14 unique
356 letters. A hyphen (-) character is used to indicate that the permission at
357 that position is not specified.
358 .sp
359 .ne 2
360 .na
361 \fB\fBba\fR\fR
362 .ad
363 .RS 5n
364 read attributes
365 .RE

367 .sp
368 .ne 2
369 .na
370 \fB\fBBA\fR\fR
371 .ad
372 .RS 5n
373 write attributes
374 .RE

376 .sp
377 .ne 2
378 .na
379 \fB\fBbc\fR\fR
380 .ad
381 .RS 5n
382 read ACL
383 .RE

385 .sp
386 .ne 2
387 .na

```

```

388 \fB\fBC\fR\fR
389 .ad
390 .RS 5n
391 write ACL
392 .RE

394 .sp
395 .ne 2
396 .na
397 \fB\fBd\fR\fR
398 .ad
399 .RS 5n
400 delete
401 .RE

403 .sp
404 .ne 2
405 .na
406 \fB\fBD\fR\fR
407 .ad
408 .RS 5n
409 delete child
410 .RE

412 .sp
413 .ne 2
414 .na
415 \fB\fBo\fR\fR
416 .ad
417 .RS 5n
418 write owner
419 .RE

421 .sp
422 .ne 2
423 .na
424 \fB\fBp\fR\fR
425 .ad
426 .RS 5n
427 append
428 .RE

430 .sp
431 .ne 2
432 .na
433 \fB\fBr\fR\fR
434 .ad
435 .RS 5n
436 read_data
437 .RE

439 .sp
440 .ne 2
441 .na
442 \fB\fBR\fR\fR
443 .ad
444 .RS 5n
445 read named attributes
446 .RE

448 .sp
449 .ne 2
450 .na
451 \fB\fBs\fR\fR
452 .ad
453 .RS 5n

```

```

454 synchronize
455 .RE

457 .sp
458 .ne 2
459 .na
460 \fB\fBw\fR\fR
461 .ad
462 .RS 5n
463 write_data
464 .RE

466 .sp
467 .ne 2
468 .na
469 \fB\fBW\fR\fR
470 .ad
471 .RS 5n
472 write named attributes
473 .RE

475 .sp
476 .ne 2
477 .na
478 \fB\fBx\fR\fR
479 .ad
480 .RS 5n
481 execute
482 .RE

484 .sp
485 .LP
486 This format allows compact permissions to be represented as, for example:
487 \fB\fBrw--d-a-----\fR
488 .sp
489 .LP
490 The fourth field is optional when \fBACL_COMPACT_FMT\fR is not specified, in
491 which case the field will be present only when the ACL entry has inheritance
492 flags set. The following is the list of inheritance flags separated by a slash
493 (/) character.
494 .sp
495 .ne 2
496 .na
497 \fB\fBdir_inherit\fR\fR
498 .ad
499 .RS 16n
500 \fBACE_DIRECTORY_INHERIT_ACE\fR
501 .RE

503 .sp
504 .ne 2
505 .na
506 \fB\fBfile_inherit\fR\fR
507 .ad
508 .RS 16n
509 \fBACE_FILE_INHERIT_ACE\fR
510 .RE

512 .sp
513 .ne 2
514 .na
515 \fB\fBinherit_only\fR\fR
516 .ad
517 .RS 16n
518 \fBACE_INHERIT_ONLY_ACE\fR
519 .RE

```

```

521 .sp
522 .ne 2
523 .na
524 \fB\fBno_propagate\fR\fR
525 .ad
526 .RS 16n
527 \fBACE_NO_PROPAGATE_INHERIT_ACE\fR
528 .RE

530 .sp
531 .LP
532 When \fBACL_COMPACT_FMT\fR is specified the inheritance will always be present
533 and is represented as positional arguments. A hyphen (-) character is used to
534 indicate that the inheritance flag at that position is not specified.
535 .sp
536 .ne 2
537 .na
538 \fB\fBd\fR\fR
539 .ad
540 .RS 5n
541 \fBdir_inherit\fR
542 .RE

544 .sp
545 .ne 2
546 .na
547 \fB\fBf\fR\fR
548 .ad
549 .RS 5n
550 \fBfile_inherit\fR
551 .RE

553 .sp
554 .ne 2
555 .na
556 \fB\fBF\fR\fR
557 .ad
558 .RS 5n
559 failed access (not currently supported)
560 .RE

562 .sp
563 .ne 2
564 .na
565 \fB\fBi\fR\fR
566 .ad
567 .RS 5n
568 \fBinherit_only\fR
569 .RE

571 .sp
572 .ne 2
573 .na
574 \fB\fBn\fR\fR
575 .ad
576 .RS 5n
577 \fBno_propagate\fR
578 .RE

580 .sp
581 .ne 2
582 .na
583 \fB\fBS\fR\fR
584 .ad
585 .RS 5n

```

```

586 successful access (not currently supported)
587 .RE

589 .sp
590 .LP
591 The fifth field contains the type of the ACE (\fBallow\fR or \fBdeny\fR):
592 .sp
593 .ne 2
594 .na
595 \fB\fBallow\fR\fR
596 .ad
597 .RS 9n
598 The mask specified in field three should be allowed.
599 .RE

601 .sp
602 .ne 2
603 .na
604 \fB\fBdeny\fR\fR
605 .ad
606 .RS 9n
607 The mask specified in field three should be denied.
608 .RE

610 .SH RETURN VALUES
614 .sp
615 .LP
611 Upon successful completion, the \fBacl_totext()\fR function returns a pointer
612 to a text string. Otherwise, it returns \fBINULL\fR.
613 .sp
614 .LP
615 Upon successful completion, the \fBacl_fromtext()\fR function returns 0.
616 Otherwise, the return value is set to one of the following:
617 .sp
618 .ne 2
619 .na
620 \fB\fBEBEACL_FIELD_NOT_BLANK\fR\fR
621 .ad
622 .RS 28n
623 A field that should be blank is not blank.
624 .RE

626 .sp
627 .ne 2
628 .na
629 \fB\fBEBEACL_FLAGS_ERROR\fR\fR
630 .ad
631 .RS 28n
632 An invalid ACL flag was specified.
633 .RE

635 .sp
636 .ne 2
637 .na
638 \fB\fBEBEACL_INHERIT_ERROR\fR\fR
639 .ad
640 .RS 28n
641 An invalid inheritance field was specified.
642 .RE

644 .sp
645 .ne 2
646 .na
647 \fB\fBEBEACL_INVALID_ACCESS_TYPE\fR\fR
648 .ad
649 .RS 28n

```

```

650 An invalid access type was specified.
651 .RE

653 .sp
654 .ne 2
655 .na
656 \fB\fBEACL_INVALID_STR\fR\fR
657 .ad
658 .RS 28n
659 The string is \fINULL\fR.
660 .RE

662 .sp
663 .ne 2
664 .na
665 \fB\fBEACL_INVALID_USER_GROUP\fR\fR
666 .ad
667 .RS 28n
668 The required user or group name not found.
669 .RE

671 .sp
672 .ne 2
673 .na
674 \fB\fBEACL_MISSING_FIELDS\fR\fR
675 .ad
676 .RS 28n
677 The ACL needs more fields to be specified.
678 .RE

680 .sp
681 .ne 2
682 .na
683 \fB\fBEACL_PERM_MASK_ERROR\fR\fR
684 .ad
685 .RS 28n
686 The permission mask is invalid.
687 .RE

689 .sp
690 .ne 2
691 .na
692 \fB\fBEACL_UNKNOWN_DATA\fR\fR
693 .ad
694 .RS 28n
695 Unknown data was found in the ACL.
696 .RE

698 .SH EXAMPLES
704 .LP
699 \fBExample 1\fR \fRExamples of permissions when \fBACL_COMPACT_FMT\fR is not
700 specified.
701 .sp
702 .in +2
703 .nf
704 user:joe:read_data/write_data:file_inherit/dir_inherit:allow
705 .fi
706 .in -2
707 .sp

709 .sp
710 .in +2
711 .nf
712 owner@:read_acl:allow,user:tom:read_data:file_inherit/inherit_only:deny
713 .fi
714 .in -2

```

```

715 .sp

717 .LP
718 \fBExample 2\fR \fRExamples of permissions when \fBACL_COMPACT_FMT\fR is
719 specified.
720 .sp
721 .in +2
722 .nf
723 user:joe:rw-----:fd----:allow
724 .fi
725 .in -2
726 .sp

728 .sp
729 .in +2
730 .nf
731 owner@:-----c---:-----allow,user:tom:r-----:f-i---:deny
732 .fi
733 .in -2
734 .sp

736 .SH ATTRIBUTES
743 .sp
744 .LP
737 See \fBattributes\fR(5) for descriptions of the following attributes:
738 .sp

740 .sp
741 .TS
742 box;
743 c | c
744 l | l .
745 ATTRIBUTE TYPE ATTRIBUTE VALUE
746 _
747 Interface Stability Committed
748 _
749 MT-Level Safe
750 .TE

752 .SH SEE ALSO
761 .sp
762 .LP
753 \fBbls\fR(1), \fBbtar\fR(1), \fBbacl\fR(2), \fBbmalloc\fR(3C),
754 \fBbaclfromtext\fR(3SEC), \fBbacl\fR(5), \fBattributes\fR(5)

```

```

*****
4759 Sat Feb 8 11:08:23 2020
new/usr/src/man/man3sec/aclcheck.3sec
12288 getfacl and setfacl could stand improvement
*****
1 \" te
2 .\" Copyright (c) 2001, Sun Microsystems, Inc.
3 .\" The contents of this file are subject to the terms of the Common Development
4 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
5 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
6 .TH ACLCHECK 3SEC "Dec 10, 2001"
7 .SH NAME
8 aclcheck - check the validity of an ACL
9 .SH SYNOPSIS
10 .LP
10 .nf
11 \fBcc\fR [ \fIflag\fR... ] \fIfile\fR... \fB-lsec\fR [ \fIlibrary\fR... ]
12 #include <sys/acl.h>

14 \fBint\fR \fBaclcheck\fR(\fBaclent_t * \fR \fIaclbufp\fR, \fBint\fR \fInentries\fR
15 .fi

17 .SH DESCRIPTION
19 .sp
20 .LP
18 The \fBaclcheck()\fR function checks the validity of an \fBACL\fR pointed to by
19 \fIaclbufp\fR. The \fInentries\fR argument is the number of entries contained
20 in the buffer. The \fIwhich\fR parameter returns the index of the first entry
21 that is invalid.
22 .sp
23 .LP
24 The function verifies that an \fBACL\fR pointed to by \fIaclbufp\fR is valid
25 according to the following rules:
26 .RS +4
27 .TP
28 .ie t \(\bu
29 .el o
30 There must be exactly one \fBGROUP_OBJ\fR \fBACL\fR entry.
31 .RE
32 .RS +4
33 .TP
34 .ie t \(\bu
35 .el o
36 There must be exactly one \fBUSER_OBJ\fR \fBACL\fR entry.
37 .RE
38 .RS +4
39 .TP
40 .ie t \(\bu
41 .el o
42 There must be exactly one \fBOTHER_OBJ\fR \fBACL\fR entry.
43 .RE
44 .RS +4
45 .TP
46 .ie t \(\bu
47 .el o
48 If there are any \fBGROUP\fR \fBACL\fR entries, then the group \fBID\fR in each
49 group \fBACL\fR entry must be unique.
50 .RE
51 .RS +4
52 .TP
53 .ie t \(\bu
54 .el o
55 If there are any \fBUSER\fR \fBACL\fR entries, then the user \fBID\fR in each
56 user \fBACL\fR entry must be unique.
57 .RE
58 .RS +4

```

```

59 .TP
60 .ie t \(\bu
61 .el o
62 If there are any \fBGROUP\fR or \fBUSER\fR \fBACL\fR entries, then there must
63 be exactly one \fBCLASS_OBJ\fR (\fBACL\fR mask) entry.
64 .RE
65 .RS +4
66 .TP
67 .ie t \(\bu
68 .el o
69 If there are any default \fBACL\fR entries, then the following apply:
70 .RS +4
71 .TP
72 .ie t \(\bu
73 .el o
74 There must be exactly one default \fBGROUP_OBJ\fR \fBACL\fR entry.
75 .RE
76 .RS +4
77 .TP
78 .ie t \(\bu
79 .el o
80 There must be exactly one default \fBOTHER_OBJ\fR \fBACL\fR entry.
81 .RE
82 .RS +4
83 .TP
84 .ie t \(\bu
85 .el o
86 There must be exactly one default \fBUSER_OBJ\fR \fBACL\fR entry.
87 .RE
88 .RS +4
89 .TP
90 .ie t \(\bu
91 .el o
92 If there are any \fBDEF_GROUP\fR entries, then the group \fBID\fR in each
93 \fBDEF_GROUP\fR \fBACL\fR entry must be unique.
94 .RE
95 .RS +4
96 .TP
97 .ie t \(\bu
98 .el o
99 If there are any \fBDEF_USER\fR entries, then the user \fBID\fR in each
100 \fBDEF_USER\fR \fBACL\fR entry must be unique.
101 .RE
102 .RS +4
103 .TP
104 .ie t \(\bu
105 .el o
106 If there are any \fBDEF_GROUP\fR or \fBDEF_USER\fR entries, then there must be
107 exactly one \fBDEF_CLASS_OBJ\fR (default \fBACL\fR mask) entry.
108 .RE
109 .RE
110 .RS +4
111 .TP
112 .ie t \(\bu
113 .el o
114 If any of the above rules are violated, then the function fails with
115 \fBerrno\fR set to \fBEINVAL\fR.
116 .RE
117 .SH RETURN VALUES
118 If the \fBACL\fR is valid, \fBaclcheck()\fR will return \fB0\fR. Otherwise
119 \fBerrno\fR is set to \fBEINVAL\fR and \fBaclcheck()\fR will return one of the
121 .sp
122 .LP
123 If the \fBACL\fR is valid, \fBaclcheck()\fR will return \fB0\fR. Otherwise
124 \fBerrno\fR is set to \fBEINVAL\fR and return code is set to one of the
120 following:

```

```

121 .sp
122 .ne 2
123 .na
124 \fB\fBGRP_ERROR\fR\fR
125 .ad
126 .RS 19n
127 There is more than one \fBGROUP_OBJ\fR or \fBDEF_GROUP_OBJ\fR \fBACL\fR entry.
128 .RE

```

```

130 .sp
131 .ne 2
132 .na
133 \fB\fBUSER_ERROR\fR\fR
134 .ad
135 .RS 19n
136 There is more than one \fBUSER_OBJ\fR or \fBDEF_USER_OBJ\fR \fBACL\fR entry.
137 .RE

```

```

139 .sp
140 .ne 2
141 .na
142 \fB\fBCLASS_ERROR\fR\fR
143 .ad
144 .RS 19n
145 There is more than one \fBCLASS_OBJ\fR (\fBACL\fR mask) or \fBDEF_CLASS_OBJ\fR
146 (default \fBACL\fR mask) entry.
147 .RE

```

```

149 .sp
150 .ne 2
151 .na
152 \fB\fBOTHER_ERROR\fR\fR
153 .ad
154 .RS 19n
155 There is more than one \fBOTHER_OBJ\fR or \fBDEF_OTHER_OBJ\fR \fBACL\fR entry.
156 .RE

```

```

158 .sp
159 .ne 2
160 .na
161 \fB\fBDUPLICATE_ERROR\fR\fR
162 .ad
163 .RS 19n
164 Duplicate entries of \fBUSER\fR, \fBGROUP\fR, \fBDEF_USER\fR, or
165 \fBDEF_GROUP\fR.
166 .RE

```

```

168 .sp
169 .ne 2
170 .na
171 \fB\fBENTRY_ERROR\fR\fR
172 .ad
173 .RS 19n
174 The entry type is invalid.
175 .RE

```

```

177 .sp
178 .ne 2
179 .na
180 \fB\fBMISS_ERROR\fR\fR
181 .ad
182 .RS 19n
183 Missing an entry. The \fIwhich\fR parameter returns \fB(mil\fR in this case.
184 .RE

```

```

186 .sp

```

```

187 .ne 2
188 .na
189 \fB\fBMEM_ERROR\fR\fR
190 .ad
191 .RS 19n
192 The system cannot allocate any memory. The \fBwhich\fR parameter returns
193 \fB(mil\fR in this case.
194 .RE

```

```

196 .SH ATTRIBUTES
202 .sp
203 .LP
197 See \fBattributes\fR(5) for descriptions of the following attributes:
198 .sp

```

```

200 .sp
201 .TS
202 box;
203 c | c
204 l | l .
205 ATTRIBUTE TYPE ATTRIBUTE VALUE
206 _
207 Interface Stability Evolving
208 _
209 MT-Level Unsafe
210 .TE

```

```

212 .SH SEE ALSO
220 .sp
221 .LP
213 \fBacl\fR(2), \fBaclsort\fR(3SEC), \fBattributes\fR(5)

```

\*\*\*\*\*

5497 Sat Feb 8 11:08:23 2020

new/usr/src/man/man3sec/acltotext.3sec

12288 getfacl and setfacl could stand improvement

\*\*\*\*\*

```

1  \' te
2  .\" Copyright (c) 2001, Sun Microsystems, Inc.
3  .\" The contents of this file are subject to the terms of the Common Development
4  .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
5  .\" When distributing Covered Code, include this CDDL HEADER in each file and in
6  .TH ACLTOTEXT 3SEC \"Dec 10, 2001\"
7  .SH NAME
8  acltotext, aclfromtext \- convert internal representation to or from external
9  representation
10 .SH SYNOPSIS
11 .LP
12 \fBcc\fR [ \fIfIflag\fR... ] \fIfIfile\fR... \fBlsec\fR [ \fIfIlibrary\fR... ]
13 #include <sys/acl.h>

15 \fBchar *\fR\fBacltotext\fR(\fBaclent_t *\fR\fIaclbufp\fR, \fBint\fR \fIaclcnt\fR
16 .fi

18 .LP
19 .nf
20 \fBaclent_t *\fR\fBaclfromtext\fR(\fBchar *\fR\fIacltextp\fR, \fBint *\fR\fIaclc
21 .fi

23 .SH DESCRIPTION
25 .sp
26 .LP
27 The \fBacltotext()\fR function converts an internal \fBACL\fR representation
28 pointed to by \fIaclbufp\fR into an external \fBACL\fR representation. The
29 space for the external text string is obtained using \fBmalloc\fR(3C). The
30 caller is responsible for freeing the space upon completion.
31 caller is responsible for freeing the space upon completion..
32 .sp
33 .LP
34 The \fBaclfromtext()\fR function converts an external \fBACL\fR representation
35 pointed to by \fIacltextp\fR into an internal \fBACL\fR representation. The
36 space for the list of \fBACL\fR entries is obtained using \fBmalloc\fR(3C). The
37 caller is responsible for freeing the space upon completion. The \fIaclcnt\fR
38 argument indicates the number of \fBACL\fR entries found.
39 .sp
40 .LP
41 An external \fBACL\fR representation is defined as follows:
42 .sp
43 .LP
44 <acl_entry>[,<acl_entry>]|\.|.|.
45 .sp
46 .LP
47 Each <acl_entry> contains one \fBACL\fR entry. The external representation of
48 an \fBACL\fR entry contains two or three colon-separated fields. The first
49 field contains the \fBACL\fR entry tag type. The entry type keywords are
50 defined as:
51 .sp
52 .ne 2
53 .na
54 \fB\fBuser\fR\fR
55 This \fBACL\fR entry with no \fBUID\fR specified in the \fBACL\fR entry
56 \fBID\fR field specifies the access granted to the owner of the object.
57 Otherwise, this \fBACL\fR entry specifies the access granted to a specific
58 user-name or user-id number.
59 .RE

```

```

59 .sp
60 .ne 2
61 .na
62 \fB\fBgroup\fR\fR
63 .ad
64 .RS 17n
65 This \fBACL\fR entry with no \fBGID\fR specified in the \fBACL\fR entry
66 \fBID\fR field specifies the access granted to the owning group of the object.
67 Otherwise, this \fBACL\fR entry specifies the access granted to a specific
68 group-name or group-id number.
69 .RE

71 .sp
72 .ne 2
73 .na
74 \fB\fBother\fR\fR
75 .ad
76 .RS 17n
77 This \fBACL\fR entry specifies the access granted to any user or group that
78 does not match any other \fBACL\fR entry.
79 .RE

81 .sp
82 .ne 2
83 .na
84 \fB\fBmask\fR\fR
85 .ad
86 .RS 17n
87 This \fBACL\fR entry specifies the maximum access granted to user or group
88 entries.
89 .RE

91 .sp
92 .ne 2
93 .na
94 \fB\fBdefault:user\fR\fR
95 .ad
96 .RS 17n
97 This \fBACL\fR entry with no uid specified in the \fBACL\fR entry \fBID\fR
98 field specifies the default access granted to the owner of the object.
99 Otherwise, this \fBACL\fR entry specifies the default access granted to a
100 specific user-name or user-\fBID\fR number.
101 .RE

103 .sp
104 .ne 2
105 .na
106 \fB\fBdefault:group\fR\fR
107 .ad
108 .RS 17n
109 This \fBACL\fR entry with no gid specified in the \fBACL\fR entry \fBID\fR
110 field specifies the default access granted to the owning group of the object.
111 Otherwise, this \fBACL\fR entry specifies the default access granted to a
112 specific group-name or group-\fBID\fR number.
113 .RE

115 .sp
116 .ne 2
117 .na
118 \fB\fBdefault:other\fR\fR
119 .ad
120 .RS 17n
121 This \fBACL\fR entry specifies the default access for other entry.
122 .RE

```

```

124 .sp
125 .ne 2
126 .na
127 \fB\fBdefault:mask\fR\fR
128 .ad
129 .RS 17n
130 This \fBACL\fR entry specifies the default access for mask entry.
131 .RE

133 .sp
134 .LP
135 The second field contains the \fBACL\fR entry \fBID\fR, as follows:
136 .sp
137 .ne 2
138 .na
139 \fB\fBuid\fR\fR
140 .ad
141 .RS 9n
142 This field specifies a user-name, or user-\fBID\fR if there is no user-name
143 associated with the user-\fBID\fR number.
144 .RE

146 .sp
147 .ne 2
148 .na
149 \fB\fBgid\fR\fR
150 .ad
151 .RS 9n
152 This field specifies a group-name, or group-\fBID\fR if there is no group-name
153 associated with the group-\fBID\fR number.
154 .RE

156 .sp
157 .ne 2
158 .na
159 \fB\fBempty\fR\fR
160 .ad
161 .RS 9n
162 This field is used by the user and group \fBACL\fR entry types.
163 .RE

165 .sp
166 .LP
167 The third field contains the following symbolic discretionary access
168 permissions:
169 .sp
170 .ne 2
171 .na
172 \fB\fBr\fR\fR
173 .ad
174 .RS 9n
175 read permission
176 .RE

178 .sp
179 .ne 2
180 .na
181 \fB\fBw\fR\fR
182 .ad
183 .RS 9n
184 write permission
185 .RE

187 .sp
188 .ne 2
189 .na

```

```

190 \fB\fBx\fR\fR
191 .ad
192 .RS 9n
193 execute/search permission
194 .RE

196 .sp
197 .ne 2
198 .na
199 \fB\fB(mi\fR \fR)
200 .ad
201 .RS 9n
202 no access
203 .RE

205 .SH RETURN VALUES
209 .sp
210 .LP
206 Upon successful completion, the \fBacltotext()\fR function returns a pointer to
207 a text string. Otherwise, it returns \fBNUL\fR.
208 .sp
209 .LP
210 Upon successful completion, the \fBaclfromtext()\fR function returns a pointer
211 to a list of \fBACL\fR entries. Otherwise, it returns \fBNUL\fR.
212 .SH ATTRIBUTES
218 .sp
219 .LP
213 See \fBattributes\fR(5) for descriptions of the following attributes:
214 .sp

216 .sp
217 .TS
218 box:
219 c | c
220 l | l
221 ATTRIBUTE TYPE ATTRIBUTE VALUE
222 -
223 Interface Stability Evolving
224 -
225 MT-Level Unsafe
226 .TE

228 .SH SEE ALSO
236 .sp
237 .LP
229 \fBacl\fR(2), \fBmalloc\fR(3C), \fBattributes\fR(5)

```

```

*****
17758 Sat Feb  8 11:08:23 2020
new/usr/src/man/man5/acl.5
12288 getfacl and setfacl could stand improvement
*****
1 \" te
2 .\" Copyright (c) 2020 Peter Tribble.
3 .\" Copyright 2014 Nexenta Systems, Inc. All rights reserved.
4 .\" Copyright (c) 2008, Sun Microsystems, Inc. All Rights Reserved.
5 .\" The contents of this file are subject to the terms of the Common Development
6 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
7 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
8 .TH ACL 5 "Feb 8, 2020"
9 .TH ACL 5 "Nov 24, 2014"
10 .SH NAME
11 acl \- Access Control Lists
12 .SH DESCRIPTION
13 .LP
14 Access control lists (ACLs) are discretionary access control mechanisms that
15 grant and deny access to files and directories. Two different ACL models are
16 supported in this release: POSIX-draft ACLs and NFSv4 ACLs.
17 supported in the Solaris release: POSIX-draft ACLs and NFSv4 ACLs.
18 .sp
19 .LP
20 The older, POSIX-draft model is supported by the UFS file system. This model is
21 based on a withdrawn ACL POSIX specification that was never standardized. It
22 was subsequently withdrawn by the POSIX committee.
23 .sp
24 .LP
25 The other model is based on the standards of the NFSv4 working group and is an
26 approved standard from the Internet Engineering Task Force (IETF). The ZFS file
27 system uses the NFSv4 model, and provides richer semantics and finer grained
28 permission capabilities than the POSIX-draft model.
29 .SS "POSIX-draft ACLs"
30 .SS "\fBPOSIX\fR-draft \fBACL\fRs"
31 .LP
32 POSIX-draft ACLs provide an alternative security mechanism to basic UNIX file
33 permissions. Their purpose is to further restrict access
34 permissions in the Solaris release. Their purpose is to further restrict access
35 to files and directories or to extend permissions to a particular user. ACLs
36 can be used to change the permissions for the standard owner, group and other
37 class bits of a file's mode. ACLs can give additional users and groups access
38 to the file. A directory can also have a special kind of ACL called a
39 \fBdefault\fR ACL, which defines ACL entries to be inherited by descendents of
40 the directory. POSIX-draft ACLs have an ACL entry called \fBmask\fR. The mask
41 defines the maximum permissions that can be granted to additional user and
42 group entries. Whenever a file is created or its mode is changed by
43 \fBchmod\fR(1) or \fBchmod\fR(2), the mask is recomputed. It is recomputed to
44 be the group permission defined in the mode passed to \fBchmod\fR(2).
45 .sp
46 .LP
47 The POSIX-draft ACL model uses the standard \fBbrwx\fR model of traditional UNIX
48 permissions.
49 .sp
50 .LP
51 An ACL is represented as follows:
52 .sp
53 .in +2
54 .nf
55 \fIacl_entry\fR[\fIacl_entry\fR]...
56 .fi
57 .in -2
58 .sp
59 .sp
60 .LP
61 .sp
62 .LP

```

```

56 Each \fIacl_entry\fR contains one ACL entry. An ACL entry is represented by two
57 or three colon-separated(\fB:\fR) fields.
58 .sp
59 .ne 2
60 .na
61 \fB\fIuser\fR:\fIuid\fR:\fIperms\fR\fR
62 .ad
63 .RS 2ln
64 If \fIuid\fR blank, it represents the file owner.
65 .RE
66 .sp
67 .ne 2
68 .na
69 .na
70 \fB\fIgroup\fR:\fIgid\fR:\fIperms\fR\fR
71 .ad
72 .RS 2ln
73 If \fIgid\fR is blank, it represents the owning group.
74 .RE
75 .sp
76 .ne 2
77 .na
78 .na
79 \fB\fIother\fR:\fIperms\fR\fR
80 .ad
81 .RS 2ln
82 Represents the file other class.
83 .RE
84 .sp
85 .ne 2
86 .na
87 .na
88 \fB\fImask\fR:\fIperms\fR\fR
89 .ad
90 .RS 2ln
91 Defines the \fBMAX\fR permission to hand out.
92 .RE
93 .sp
94 .LP
95 For example to give user \fBjoe\fR read and write permissions, the ACL entry is
96 specified as:
97 .sp
98 .in +2
99 .nf
100 user:joe:rw-
101 .fi
102 .in -2
103 .sp
104 .sp
105 .SS "NFSv4 ACLs"
106 The NFSv4 ACL model is based loosely on the Windows NT ACL model. NFSv4 ACLs
107 .SS "\fBNFS\fRv4 \fBACL\fRs"
108 .LP
109 NFSv4 ACL model is based loosely on the Windows NT ACL model. NFSv4 ACLs
110 provide a much richer ACL model than POSIX-draft ACLs.
111 .sp
112 .LP
113 The major differences between NFSv4 and POSIX-draft ACLs are as follows:
114 .RS +4
115 .ie t \(\bu
116 .el o
117 NFSv4 ACLs provide finer grained permissions than the \fBbrwx\fR model.
118 .RE +4

```

```

119 .TP
120 .ie t \(\bu
121 .el o
122 NFSv4 ACLs allow for both \fBALLOW\fR and \fBDENY\fR entries.
123 .RE
124 .RS +4
125 .TP
126 .ie t \(\bu
127 .el o
128 NFSv4 ACLs provide a rich set of inheritance semantics. POSIX ACLs also have
129 inheritance, but with the NFSv4 model you can control the following inheritance
130 features:
131 .RS +4
132 .TP
133 .ie t \(\bu
134 .el o
135 Whether inheritance cascades to both files and directories or only to files or
136 directories.
137 .RE
138 .RS +4
139 .TP
140 .ie t \(\bu
141 .el o
142 In the case of directories, you can indicate whether inheritance is applied to
143 the directory itself, to just one level of subdirectories, or cascades to all
144 subdirectories of the directory.
145 .RE
146 .RE
147 .RS +4
148 .TP
149 .ie t \(\bu
150 .el o
151 NFSv4 ACLs provide a mechanism for hooking into a system's audit trail.
152 Currently, illumos does not support this mechanism.
153 Currently, Solaris does not support this mechanism.
154 .RE
155 .RS +4
156 .TP
157 .ie t \(\bu
158 .el o
159 NFSv4 ACLs enable administrators to specify the order in which ACL entries are
160 checked. With POSIX-draft ACLs the file system reorders ACL entries into a well
161 defined, strict access, checking order.
162 .sp
163 .LP
164 POSIX-draft ACL semantics can be achieved with NFSv4 ACLs. However, only some
165 NFSv4 ACLs can be translated to equivalent POSIX-draft ACLs.
166 .sp
167 .LP
168 Permissions can be specified in three different \fBchmod\fR ACL formats:
169 verbose, compact, or positional. The verbose format uses words to indicate that
170 the permissions are separated with a forward slash (\fB/\fR) character. Compact
171 format uses the permission letters and positional format uses the permission
172 letters or the hyphen (\fB-\fR) to identify no permissions.
173 .sp
174 .LP
175 The permissions for verbose mode and their abbreviated form in parentheses for
176 compact and positional mode are described as follows:
177 .sp
178 .ne 2
179 .na
180 \fBread_data (\fBr\fR)\fR
181 .ad
182 .RS 24n
183 Permission to read the data of the file

```

```

184 .RE
186 .sp
187 .ne 2
188 .na
189 \fBlist_directory (\fBr\fR)\fR
190 .ad
191 .RS 24n
192 Permission to list the contents of a directory.
193 .RE
195 .sp
196 .ne 2
197 .na
198 \fBwrite_data (\fBw\fR)\fR
199 .ad
200 .RS 24n
201 Permission to modify a file's data anywhere in the file's offset range. This
202 includes the ability to grow the file or write to any arbitrary offset.
203 .RE
205 .sp
206 .ne 2
207 .na
208 \fBadd_file (\fBw\fR)\fR
209 .ad
210 .RS 24n
211 Permission to add a new file to a directory.
212 .RE
214 .sp
215 .ne 2
216 .na
217 \fBappend_data (\fBp\fR)\fR
218 .ad
219 .RS 24n
220 The ability to modify the file's data, but only starting at EOF. Currently,
221 this permission is not supported.
222 .RE
224 .sp
225 .ne 2
226 .na
227 \fBadd_subdirectory (\fBp\fR)\fR
228 .ad
229 .RS 24n
230 Permission to create a subdirectory to a directory.
231 .RE
233 .sp
234 .ne 2
235 .na
236 \fBread_xattr (\fBR\fR)\fR
237 .ad
238 .RS 24n
239 The ability to read the extended attributes of a file or do a lookup in the
240 extended attributes directory.
241 .RE
243 .sp
244 .ne 2
245 .na
246 \fBwrite_xattr (\fBW\fR)\fR
247 .ad
248 .RS 24n
249 The ability to create extended attributes or write to the extended attributes

```

```

250 directory.
251 .RE

253 .sp
254 .ne 2
255 .na
256 \fBexecute (\fBx\fR)\fR
257 .ad
258 .RS 24n
259 Permission to execute a file.
260 .RE

262 .sp
263 .ne 2
264 .na
265 \fBread_attributes (\fBa\fR)\fR
266 .ad
267 .RS 24n
268 The ability to read basic attributes (non-ACLs) of a file. Basic attributes are
269 considered to be the stat level attributes. Allowing this access mask bit means
270 that the entity can execute \fBl\s\fR(1) and \fBstat\fR(2).
271 .RE

273 .sp
274 .ne 2
275 .na
276 \fBwrite_attributes (\fBA\fR)\fR
277 .ad
278 .RS 24n
279 Permission to change the times associated with a file or directory to an
280 arbitrary value.
281 .RE

283 .sp
284 .ne 2
285 .na
286 \fBdelete (\fBd\fR)\fR
287 .ad
288 .RS 24n
289 Permission to delete the file.
290 .RE

292 .sp
293 .ne 2
294 .na
295 \fBdelete_child (\fBD\fR)\fR
296 .ad
297 .RS 24n
298 Permission to delete a file within a directory.
299 .RE

301 .sp
302 .ne 2
303 .na
304 \fBread_acl (\fBc\fR)\fR
305 .ad
306 .RS 24n
307 Permission to read the ACL.
308 .RE

310 .sp
311 .ne 2
312 .na
313 \fBwrite_acl (\fBC\fR)\fR
314 .ad
315 .RS 24n

```

```

316 Permission to write the ACL or the ability to execute \fBchmod\fR(1) or
317 \fBsetfacl\fR(1).
318 .RE

320 .sp
321 .ne 2
322 .na
323 \fBwrite_owner (\fBo\fR)\fR
324 .ad
325 .RS 24n
326 Permission to change the owner or the ability to execute \fBchown\fR(1) or
327 \fBchgrp\fR(1).
328 .RE

330 .sp
331 .ne 2
332 .na
333 \fBsynchronize (\fBs\fR)\fR
334 .ad
335 .RS 24n
336 Permission to access a file locally at the server with synchronous reads and
337 writes. Currently, this permission is not supported.
338 .RE

340 .sp
341 .LP
342 The following inheritance flags are supported by NFSv4 ACLs:
343 .sp
344 .ne 2
345 .na
346 \fBfile_inherit (\fBF\fR)\fR
347 .ad
348 .RS 26n
349 Inherit to all newly created files in a directory.
350 .RE

352 .sp
353 .ne 2
354 .na
355 \fBdir_inherit (\fBd\fR)\fR
356 .ad
357 .RS 26n
358 Inherit to all newly created directories in a directory.
359 .RE

361 .sp
362 .ne 2
363 .na
364 \fBinherit_only (\fBi\fR)\fR
365 .ad
366 .RS 26n
367 Placed on a directory, but does not apply to the directory itself, only to
368 newly created files and directories. This flag requires file_inherit
369 and/or dir_inherit to indicate what to inherit.
370 .RE

372 .sp
373 .ne 2
374 .na
375 \fBno_propagate (\fBn\fR)\fR
376 .ad
377 .RS 26n
378 Placed on directories and indicates that ACL entries should only be inherited
379 one level of the tree. This flag requires file_inherit and/or dir_inherit to
380 indicate what to inherit.
381 .RE

```

```

383 .sp
384 .ne 2
385 .na
386 \fBsuccessful_access (\fBS\fR)\fR
387 .ad
388 .RS 26n
389 Indicates whether an alarm or audit record should be initiated upon successful
390 accesses. Used with audit/alarm ACE types.
391 .RE

393 .sp
394 .ne 2
395 .na
396 \fBfailed_access (\fBF\fR)\fR
397 .ad
398 .RS 26n
399 Indicates whether an alarm or audit record should be initiated when access
400 fails. Used with audit/alarm ACE types.
401 .RE

403 .sp
404 .ne 2
405 .na
406 \fBinherited (\fBI\fR)\fR
407 .ad
408 .RS 26n
409 ACE was inherited.
410 .RE

412 .sp
413 .ne 2
414 .na
415 \fB\fB-\fR\fR
416 .ad
417 .RS 26n
418 No permission granted.
419 .RE

421 .sp
422 .LP
423 An NFSv4 ACL is expressed using the following syntax:
424 .sp
425 .in +2
426 .nf
427 \fIIacl_entry\fR[, \fIIacl_entry\fR]...

429 owner@:<perms>[:inheritance flags]:<allow|deny>
430 group@:<perms>[:inheritance flags]:<allow|deny>
431 everyone@:<perms>[:inheritance flags]:<allow|deny>
432 user:<username>:<perms>[:inheritance flags]:<allow|deny>
433 usersid:<sid string>:<perms>[:inheritance flags]:<allow|deny>
434 group:<groupname>:<perms>[:inheritance flags]:<allow|deny>
435 groupsid:<sid string>:<perms>[:inheritance flags]:<allow|deny>
436 sid:<sid string>:<perms>[:inheritance flags]:<allow|deny>
437 .fi
438 .in -2

440 .sp
441 .ne 2
442 .na
443 \fBowner@\fR
444 .ad
445 .RS 10n
446 File owner
447 .RE

```

```

449 .sp
450 .ne 2
451 .na
452 \fBgroup@\fR
453 .ad
454 .RS 10n
455 Group owner
456 .RE

458 .sp
459 .ne 2
460 .na
461 \fBuser@\fR
462 .ad
463 .RS 10n
464 Permissions for a specific user
465 .RE

467 .sp
468 .ne 2
469 .na
470 \fBgroup\fR
471 .ad
472 .RS 10n
473 Permissions for a specific group
474 .RE

476 .sp
477 .LP
478 Permission and inheritance flags are separated by a \fB/\fR character.
479 .sp
480 .LP
481 ACL specification examples:
482 .sp
483 .in +2
484 .nf
485 user:fred:read_data/write_data/read_attributes:file_inherit:allow
486 owner@:read_data:allow,group@:read_data:allow,user:tom:read_data:deny
487 .fi
488 .in -2
489 .sp

491 .sp
492 .LP
493 Using the compact ACL format, permissions are specified by using 14 unique
494 letters to indicate permissions.
495 .sp
496 .LP
497 Using the positional ACL format, permissions are specified as positional
498 arguments similar to the \fBls -V\fR format. The hyphen (\fB-\fR), which
499 indicates that no permission is granted at that position, can be omitted and
500 only the required letters have to be specified.
501 .sp
502 .LP
503 The letters above are listed in the order they would be specified in positional
504 notation.
505 .sp
506 .LP
507 With these letters you can specify permissions in the following equivalent
508 ways.
509 .sp
510 .in +2
511 .nf
512 user:fred:rw-----R-----:file_inherit:allow
513 .fi

```

```

514 .in -2
515 .sp

517 .sp
518 .LP
519 Or you can remove the \fB-\fR and scrunch it together.
520 .sp
521 .in +2
522 .nf
523 user:fred:rwR:file_inherit:allow
524 .fi
525 .in -2
526 .sp

528 .sp
529 .LP
530 The inheritance flags can also be specified in a more compact manner, as
531 follows:
532 .sp
533 .in +2
534 .nf
535 user:fred:rwR:f:allow
536 user:fred:rwR:f-----:allow
537 .fi
538 .in -2
539 .sp

541 SS "Shell-level API"
542 Several utilities support the manipulation of ACLs. The following
543 utilities accommodate both ACL models:
544 SS "Shell-level Solaris \fBAPI\fR"
544 .LP
545 The Solaris command interface supports the manipulation of ACLs. The following
546 Solaris utilities accommodate both ACL models:
544 .sp
545 .ne 2
546 .na
547 \fB\fBchmod\fR\fR
548 .ad
549 .RS 12n
550 The \fBchmod\fR utility has been enhanced to allow for the setting and deleting
551 of ACLs. This is achieved by extending the symbolic-mode argument to support
552 ACL manipulation. See \fBchmod\fR(1) for details.
553 .RE

555 .sp
556 .ne 2
557 .na
558 \fB\fBcompress\fR\fR
559 .ad
560 .RS 12n
561 When a file is compressed any ACL associated with the original file is
562 preserved with the compressed file.
563 .RE

565 .sp
566 .ne 2
567 .na
568 \fB\fBcp\fR\fR
569 .ad
570 .RS 12n
571 By default, \fBcp\fR ignores ACLs, unless the \fB-p\fR option is specified.
572 When \fB-p\fR is specified the owner and group id, permission modes,
573 modification and access times, ACLs, and extended attributes if applicable are
574 preserved.
575 .RE

```

```

577 .sp
578 .ne 2
579 .na
580 \fB\fBcpio\fR\fR
581 .ad
582 .RS 12n
583 ACLs are preserved when the \fB-P\fR option is specified.
584 .RE

586 .sp
587 .ne 2
588 .na
589 \fB\fBfind\fR\fR
590 .ad
591 .RS 12n
592 Find locates files with ACLs when the \fB-acl\fR flag is specified.
593 .RE

595 .sp
596 .ne 2
597 .na
598 \fB\fBls\fR\fR
599 .ad
600 .RS 12n
601 By default \fBls\fR does not display ACL information. When the \fB-v\fR option
602 is specified, a file's ACL is displayed.
603 .RE

605 .sp
606 .ne 2
607 .na
608 \fB\fBmv\fR\fR
609 .ad
610 .RS 12n
611 When a file is moved, all attributes are carried along with the renamed file.
612 When a file is moved across a file system boundary, the ACLs are replicated. If
613 the ACL information cannot be replicated, the move fails and the source file is
614 not removed.
615 .RE

617 .sp
618 .ne 2
619 .na
620 \fB\fBpack\fR\fR
621 .ad
622 .RS 12n
623 When a file is packed, any ACL associated with the original file is preserved
624 with the packed file.
625 .RE

627 .sp
628 .ne 2
629 .na
630 \fB\fBrcp\fR\fR
631 .ad
632 .RS 12n
633 \fBrcp\fR has been enhanced to support copying. A file's ACL is only preserved
634 when the remote host supports ACLs.
635 .RE

637 .sp
638 .ne 2
639 .na
640 \fB\fBtar\fR\fR
641 .ad

```

```

642 .RS 12n
643 ACLs are preserved when the \fB-p\fR option is specified.
644 .RE

646 .sp
647 .ne 2
648 .na
649 \fB\fBunpack\fR\fR
650 .ad
651 .RS 12n
652 When a file with an ACL is unpacked, the unpacked file retains the ACL
653 information.
654 .RE

656 .SS "Application-level API"
659 .SS "Application-level \fBAPI\fR"
660 .LP
661 The primary interfaces required to access file system ACLs at the programmatic
662 level are the \fBacl_get()\fR and \fBacl_set()\fR functions. These functions
663 support both POSIX-draft ACLs and NFSv4 ACLs.
664 .SS "Retrieving a file's ACL"
665 support both POSIX draft ACLs and NFSv4 ACLs.
666 .SS "Retrieving a file's \fBACL\fR"
667 .in +2
668 .nf
669 int acl_get(const char *path, int flag, acl_t **aclp);
670 int facl_get(int fd, int flag, acl_t **aclp);
671 .fi
672 .in -2

674 .sp
675 .LP
676 The \fBacl_get()\fR(3SEC) and \fBfacl_get()\fR(3SEC) functions retrieve an ACL on
677 The \fBacl_get()\fR(3SEC) and \fBfacl_get()\fR(3SEC) functions retrieves an ACL on
678 a file whose name is given by path or referenced by the open file descriptor
679 fd. The flag argument specifies whether a trivial ACL should be retrieved. When
680 the flag argument equals \fBACL_NO_TRIVIAL\fR only ACLs that are not
681 the flag argument equals \fBACL_NO_TRIVIAL\fR then only ACLs that are not
682 trivial are retrieved. The ACL is returned in the \fBaclp\fR argument.
683 .SS "Freeing ACL structure"
684 .SS "Freeing \fBACL\fR structure"
685 .in +2
686 .nf
687 void acl_free(acl_t *aclp);
688 void facl_free(acl_t *aclp);
689 .fi
690 .in -2

692 .sp
693 .LP
694 The \fBacl_free()\fR function frees up memory allocated for the argument
695 \fBaclp\fR.
696 .SS "Setting an ACL on a file"
697 \fBaclp\fR.
698 .SS "Setting an \fBACL\fR on a file"
699 .in +2
700 .nf
701 int acl_set(const char *path, acl_t *aclp);
702 int facl_set(int fd, acl_t *aclp);
703 .fi
704 .in -2

706 .sp
707 .LP
708 The \fBacl_set()\fR(3SEC) and \fBfacl_get()\fR(3SEC) functions are used for setting
709 an ACL on a file whose name is given by path or referenced by the open file

```

```

698 descriptor \fBfd\fR. The \fBaclp\fR argument specifies the ACL to set. The
699 \fBacl_set()\fR(3SEC) function translates a POSIX-draft ACL into a NFSv4 ACL when
700 the target file system supports NFSv4 ACLs. No translation is performed when
701 \fBacl_set()\fR(3SEC) translates an POSIX-draft ACL into a NFSv4 ACL when the
702 target file systems supports NFSv4 ACLs. No translation is performed when
703 trying to set an NFSv4 ACL on a POSIX-draft ACL supported file system.
704 .SS "Determining an ACL's trivialness"
705 .SS "Determining an \fBACL\fR's trivialness"
706 .in +2
707 .nf
708 int acl_trivial(const char *path);
709 .fi
710 .in -2

712 .sp
713 .LP
714 The \fBacl_trivial()\fR function is used to determine whether a file has a
715 trivial ACL.
716 .SS "Removing all ACLs from a file"
717 .SS "Removing all \fBACL\fRs from a file"
718 .in +2
719 .nf
720 int acl_strip(const char *path, uid_t uid, gid_t gid, mode_t mode);
721 .fi
722 .in -2

724 .sp
725 .LP
726 The \fBacl_strip()\fR function removes all ACLs from a file and replaces them
727 with a trivial ACL based off of the passed in argument mode. After replacing
728 the ACL the owner and group of the file are set to the values specified in the
729 uid and gid parameters.
730 .SS "Converting ACLs to/from external representation"
731 .SS "Converting \fBACL\fRs to/from external representation"
732 .in +2
733 .nf
734 int acl_fromtext(const char *path, acl_t **aclp);
735 char *acl_totext(acl_t *aclp, int flags);
736 .fi
737 .in -2

739 .sp
740 .LP
741 The \fBacl_totext()\fR function converts an internal ACL representation pointed
742 to by aclp into an external representation. See \fBDESCRIPTION\fR for details
743 about external representation.
744 .sp
745 .LP
746 The \fBacl_fromtext()\fR function converts an external representation into an
747 internal representation. See \fBDESCRIPTION\fR for details about external
748 representation.
749 .SH EXAMPLES
750 .LP
751 The following examples demonstrate how the API can be used to perform basic
752 operations on ACLs.
753 .LP
754 \fBExample 1 \fR Retrieving and Setting an ACL
755 .sp
756 .LP
757 Use the following to retrieve an ACL and set it on another file:

759 .sp
760 .in +2
761 .nf
762 error = acl_get("file", ACL_NO_TRIVIAL, &aclp);

```

```
758 if (error == 0 && aclp != NULL) {
759 .in +8
760 error = acl_set("file2", aclp);
761 acl_free(aclp);
762 .in -8
763 }
unchanged_portion_omitted
785 \&...
786 .fi
787 .in -2

789 .LP
790 \fBExample 3 \fRDetermining if a File has a Trivial ACL
791 .sp
792 .LP
793 Use the following to determine if a file has a trivial ACL:

795 .sp
796 .in +2
797 .nf
798 char *file = "file5";
799 istrivial = acl_trivial(file);

801 if (istrivial == 0)
802 .in +8
803 printf("file %s has a trivial ACL\n", file);
804 .in -8
805 else
806 .in +8
807 printf("file %s has a NON-trivial ACL\n", file);
808 .in -8
809 \&...
810 .fi
811 .in -2

813 .LP
814 \fBExample 4 \fRRemoving all ACLs from a File
815 .sp
816 .LP
817 Use the following to remove all ACLs from a file, and set a new mode, owner,
818 and group:

820 .sp
821 .in +2
822 .nf
823 error = acl_strip("file", 10, 100, 0644);
824 \&...
825 .fi
826 .in -2

828 .SH SEE ALSO
834 .LP
829 \fBchgrp\fR(1), \fBchmod\fR(1), \fBchown\fR(1), \fBcp\fR(1), \fBcpio\fR(1),
830 \fBfind\fR(1), \fBls\fR(1), \fBmv\fR(1), \fBtar\fR(1), \fBsetfacl\fR(1),
831 \fBchmod\fR(2), \fBacl\fR(2), \fBstat\fR(2), \fBacl_get\fR(3SEC),
832 \fBaclsort\fR(3SEC), \fBacl_fromtext\fR(3SEC), \fBacl_free\fR(3SEC),
833 \fBacl_strip\fR(3SEC), \fBacl_trivial\fR(3SEC)
```