****
46313 Wed May 29 12:55:53 2013
new/usr/src/cmd/find/find.c
3795 find does not support -path or -ipath
****

```
  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) 1988, 2010, Oracle and/or its affiliates. All rights reserved.
 23  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 24  * Copyright (c) 2013 Andrew Stormont.  All rights reserved.
 25 #endif /* ! codereview */
 26  */


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


 33 /*      Parts of this product may be derived from              */
 34 /*      Mortice Kern Systems Inc. and Berkeley 4.3 BSD systems. */
 35 /*      licensed from  Mortice Kern Systems Inc. and           */
 36 /*      the University of California.                          */

 38 /*
 39  * Copyright 1985, 1990 by Mortice Kern Systems Inc.  All rights reserved.
 40  */

 42 #include <stdio.h>
 43 #include <errno.h>
 44 #include <pwd.h>
 45 #include <grp.h>
 46 #include <sys/types.h>
 47 #include <sys/stat.h>
 48 #include <sys/param.h>
 49 #include <sys/acl.h>
 50 #include <limits.h>
 51 #include <unistd.h>
 52 #include <stdlib.h>
 53 #include <locale.h>
 54 #include <string.h>
 55 #include <strings.h>
 56 #include <ctype.h>
 57 #include <wait.h>
 58 #include <fnmatch.h>
 59 #include <langinfo.h>
 60 #include <ftw.h>
 61 #include <libgen.h>
```



```
 62 #include <err.h>
 63 #include <regex.h>
 64 #include "getresponse.h"

 66 #define A_DAY           (long)(60*60*24)        /* a day full of seconds */
 67 #define A_MIN           (long)(60)
 68 #define BLKSIZ          512
 69 #define round(x, s)     (((x)+(s)-1)&~((s)-1))
 70 #ifndef FTW_SLN
 71 #define FTW_SLN         7
 72 #endif
 73 #define LINEBUF_SIZE            LINE_MAX        /* input or output lines */
 74 #define REMOTE_FS               "/etc/dfs/fstypes"
 75 #define N_FSTYPES               20
 76 #define SHELL_MAXARGS           253     /* see doexec() for description */

 78 /*
 79  * This is the list of operations
 80  * F_USER and F_GROUP are named to avoid conflict with USER and GROUP defined
 81  * in sys/acl.h
 82  */

 84 enum Command
 85 {
 86         PRINT,
 87         ACL, AMIN, AND, ATIME, CMIN, CPIO, CSIZE, CTIME, DEPTH, EXEC, F_GROUP,
 88         F_GROUPACL, F_USER, F_USERACL, FOLLOW, FSTYPE, INAME, INUM, IPATH,
 89         IREGEX, LINKS, LOCAL, LPAREN, LS, MAXDEPTH, MINDEPTH, MMIN, MOUNT,
 90         MTIME, NAME, NCPIO, NEWER, NOGRP, NOT, NOUSER, OK, OR, PATH, PERM,
 91         PRINT0, PRUNE, REGEX, RPAREN, SIZE, TYPE, VARARGS, XATTR
 24         F_GROUPACL, F_USER, F_USERACL, FOLLOW, FSTYPE, INAME, INUM, IREGEX,
 25         LINKS, LOCAL, LPAREN, LS, MAXDEPTH, MINDEPTH, MMIN, MOUNT, MTIME, NAME,
 26         NCPIO, NEWER, NOGRP, NOT, NOUSER, OK, OR, PERM, PRINT0, PRUNE, REGEX,
 27         RPAREN, SIZE, TYPE, VARARGS, XATTR
 92 };
_____unchanged_portion_omitted_

106 /*
107  * Except for pathnames, these are the only legal arguments
108  */
109 static struct Args commands[] =
110 {
111         "!",            NOT,            Op,
112         "(",            LPAREN,         Unary,
113         ")",            RPAREN,         Unary,
114         "-a",           AND,            Op,
115         "-acl",         ACL,            Unary,
116         "-amin",        AMIN,           Num,
117         "-and",         AND,            Op,
118         "-atime",       ATIME,          Num,
119         "-cmin",        CMIN,           Num,
120         "-cpio",        CPIO,           Cpio,
121         "-ctime",       CTIME,          Num,
122         "-depth",       DEPTH,          Unary,
123         "-exec",        EXEC,           Exec,
124         "-follow",      FOLLOW,         Unary,
125         "-fstype",      FSTYPE,         Str,
126         "-group",       F_GROUP,        Num,
127         "-groupacl",    F_GROUPACL,     Num,
128         "-iname",       INAME,          Str,
129         "-inum",        INUM,           Num,
130         "-ipath",       IPATH,          Str,
131 #endif /* ! codereview */
132         "-iregex",      IREGEX,         Str,
133         "-links",       LINKS,          Num,
134         "-local",       LOCAL,          Unary,
```

```
135         "-ls",          LS,             Unary,
136         "-maxdepth",    MAXDEPTH,       Num,
137         "-mindepth",    MINDEPTH,       Num,
138         "-mmin",        MMIN,           Num,
139         "-mount",       MOUNT,          Unary,
140         "-mtime",       MTIME,          Num,
141         "-name",        NAME,           Str,
142         "-ncpio",       NCPIO,          Cpio,
143         "-newer",       NEWER,          Str,
144         "-nogroup",     NOGRP,          Unary,
145         "-not",         NOT,            Op,
146         "-nouser",      NOUSER,         Unary,
147         "-o",           OR,             Op,
148         "-ok",          OK,             Exec,
149         "-or",          OR,             Op,
150         "-path",        PATH,           Str,
151 #endif /* ! codereview */
152         "-perm",        PERM,           Num,
153         "-print",       PRINT,          Unary,
154         "-print0",      PRINT0,         Unary,
155         "-prune",       PRUNE,          Unary,
156         "-regex",       REGEX,          Str,
157         "-size",        SIZE,           Num,
158         "-type",        TYPE,           Num,
159         "-user",        F_USER,         Num,
160         "-useracl",     F_USERACL,      Num,
161         "-xattr",       XATTR,          Unary,
162         "-xdev",        MOUNT,          Unary,
163         NULL,           0,              0
164 };

166 union Item
167 {
168         struct Node     *np;
169         struct Arglist  *vp;
170         time_t          t;
171         char            *cp;
172         char            **ap;
173         long            l;
174         int             i;
175         long long       ll;
176 };

178 struct Node
179 {
180         struct Node     *next;
181         enum Command    action;
182         enum Type       type;
183         union Item      first;
184         union Item      second;
185 };

187 /* if no -print, -exec or -ok replace "expression" with "(expression) -print" */
188 static  struct  Node PRINT_NODE = { 0, PRINT, 0, 0};
189 static  struct  Node LPAREN_NODE = { 0, LPAREN, 0, 0};


192 /*
193  * Prototype variable size arglist buffer
194  */

196 struct Arglist
197 {
198         struct Arglist  *next;
199         char            *end;
200         char            *nextstr;
```



```
201         char            **firstvar;
202         char            **nextvar;
203         char            *arglist[1];
204 };


207 static int              compile();
208 static int              execute();
209 static int              doexec(char *, char **, int *);
210 static struct Args      *lookup();
211 static int              ok();
212 static void             usage(void)     __NORETURN;
213 static struct Arglist   *varargs();
214 static int              list();
215 static char             *getgroup();
216 static FILE             *cmdopen();
217 static int              cmdclose();
218 static char             *getshell();
219 static void             init_remote_fs();
220 static char             *getname();
221 static int              readmode();
222 static mode_t           getmode();
223 static char             *gettail();


226 static int walkflags = FTW_CHDIR|FTW_PHYS|FTW_ANYERR|FTW_NOLOOP;
227 static struct Node      *topnode;
228 static struct Node      *freenode;      /* next free node we may use later */
229 static char             *cpio[] = { "cpio", "-o", 0 };
230 static char             *ncpio[] = { "cpio", "-oc", 0 };
231 static char             *cpiol[] = { "cpio", "-oL", 0 };
232 static char             *ncpiol[] = { "cpio", "-ocL", 0 };
233 static time_t           now;
234 static FILE             *output;
235 static char             *dummyarg = (char *)-1;
236 static int              lastval;
237 static int              varsize;
238 static struct Arglist   *lastlist;
239 static char             *cmdname;
240 static char             *remote_fstypes[N_FSTYPES+1];
241 static int              fstype_index = 0;
242 static int              action_expression = 0;  /* -print, -exec, or -ok */
243 static int              error = 0;
244 static int              paren_cnt = 0;  /* keeps track of parentheses */
245 static int              Eflag = 0;
246 static int              hflag = 0;
247 static int              lflag = 0;
248 /* set when doexec()-invoked utility returns non-zero */
249 static int              exec_exitcode = 0;
250 static regex_t          *preg = NULL;
251 static int              npreg = 0;
252 static int              mindepth = -1, maxdepth = -1;
253 extern char             **environ;

255 int
256 main(int argc, char **argv)
257 {
258         char *cp;
259         int c;
260         int paths;
261         char *cwdpath;

263         (void) setlocale(LC_ALL, "");
264 #if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
265 #define TEXT_DOMAIN "SYS_TEST"  /* Use this only if it weren't */
266 #endif
```

```
267         (void) textdomain(TEXT_DOMAIN);

269         cmdname = argv[0];
270         if (time(&now) == (time_t)(-1)) {
271                 (void) fprintf(stderr, gettext("%s: time() %s\n"),
272                     cmdname, strerror(errno));
273                 exit(1);
274         }
275         while ((c = getopt(argc, argv, "EHL")) != -1) {
276                 switch (c) {
277                 case 'E':
278                         Eflag = 1;
279                         break;
280                 case 'H':
281                         hflag = 1;
282                         lflag = 0;
283                         break;
284                 case 'L':
285                         hflag = 0;
286                         lflag = 1;
287                         break;
288                 case '?':
289                         usage();
290                         break;
291                 }
292         }

294         argc -= optind;
295         argv += optind;

297         if (argc < 1) {
298                 (void) fprintf(stderr,
299                     gettext("%s: insufficient number of arguments\n"), cmdname);
300                 usage();
301         }

303         for (paths = 0; (cp = argv[paths]) != 0; ++paths) {
304                 if (*cp == '-')
305                         break;
306                 else if ((*cp == '!' || *cp == '(') && *(cp+1) == 0)
307                         break;
308         }

310         if (paths == 0) /* no path-list */
311                 usage();

313         output = stdout;

315         /* lflag is the same as -follow */
316         if (lflag)
317                 walkflags &= ~FTW_PHYS;

319         /* allocate enough space for the compiler */
320         topnode = malloc((argc + 1) * sizeof (struct Node));
321         (void) memset(topnode, 0, (argc + 1) * sizeof (struct Node));

323         if (compile(argv + paths, topnode, &action_expression) == 0) {
324                 /* no expression, default to -print */
325                 (void) memcpy(topnode, &PRINT_NODE, sizeof (struct Node));
326         } else if (!action_expression) {
327                 /*
328                  * if no action expression, insert an LPAREN node above topnode,
329                  * with a PRINT node as its next node
330                  */
331                 struct Node *savenode;
```



```
333                 if (freenode == NULL) {
334                         (void) fprintf(stderr, gettext("%s: can't append -print"
335                             " implicitly; try explicit -print option\n"),
336                             cmdname);
337                         exit(1);
338                 }
339                 savenode = topnode;
340                 topnode = freenode++;
341                 (void) memcpy(topnode, &LPAREN_NODE, sizeof (struct Node));
342                 topnode->next = freenode;
343                 topnode->first.np = savenode;
344                 (void) memcpy(topnode->next, &PRINT_NODE, sizeof (struct Node));
345         }

347         while (paths--) {
348                 char *curpath;
349                 struct stat sb;

351                 curpath = *(argv++);

353                 /*
354                  * If -H is specified, it means we walk the first
355                  * level (pathname on command line) logically, following
356                  * symlinks, but lower levels are walked physically.
357                  * We use our own secret interface to nftw() to change
358                  * the from stat to lstat after the top level is walked.
359                  */
360                 if (hflag) {
361                         if (stat(curpath, &sb) < 0 && errno == ENOENT)
362                                 walkflags &= ~FTW_HOPTION;
363                         else
364                                 walkflags |= FTW_HOPTION;
365                 }

367                 /*
368                  * We need this check as nftw needs a CWD and we have no
369                  * way of returning back from that code with a meaningful
370                  * error related to this
371                  */
372                 if ((cwdpath = getcwd(NULL, PATH_MAX)) == NULL) {
373                         if ((errno == EACCES) && (walkflags & FTW_CHDIR)) {
374                                 /*
375                                  * A directory above cwd is inaccessible,
376                                  * so don't do chdir(2)s. Slower, but at least
377                                  * it works.
378                                  */
379                                 walkflags &= ~FTW_CHDIR;
380                                 free(cwdpath);
381                         } else {
382                                 (void) fprintf(stderr,
383                                     gettext("%s : cannot get the current "
384                                     "working directory\n"), cmdname);
385                                 exit(1);
386                         }
387                 } else
388                         free(cwdpath);


391                 if (nftw(curpath, execute, 1000, walkflags)) {
392                         (void) fprintf(stderr,
393                             gettext("%s: cannot open %s: %s\n"),
394                             cmdname, curpath, strerror(errno));
395                         error = 1;
396                 }

398         }
```

```
400         /* execute any remaining variable length lists */
401         while (lastlist) {
402                 if (lastlist->end != lastlist->nextstr) {
403                         *lastlist->nextvar = 0;
404                         (void) doexec((char *)0, lastlist->arglist,
405                             &exec_exitcode);
406                 }
407                 lastlist = lastlist->next;
408         }
409         if (output != stdout)
410                 return (cmdclose(output));
411         return ((exec_exitcode != 0) ? exec_exitcode : error);
412 }

414 /*
415  * compile the arguments
416  */

418 static int
419 compile(argv, np, actionp)
420 char **argv;
421 struct Node *np;
422 int *actionp;
423 {
424         char *b;
425         char **av;
426         struct Node *oldnp = topnode;
427         struct Args *argp;
428         char **com;
429         int i;
430         enum Command wasop = PRINT;

432         if (init_yes() < 0) {
433                 (void) fprintf(stderr, gettext(ERR_MSG_INIT_YES),
434                     strerror(errno));
435                 exit(1);
436         }

438         for (av = argv; *av && (argp = lookup(*av)); av++) {
439                 np->next = 0;
440                 np->action = argp->action;
441                 np->type = argp->type;
442                 np->second.i = 0;
443                 if (argp->type == Op) {
444                         if (wasop == NOT || (wasop && np->action != NOT)) {
445                                 (void) fprintf(stderr,
446                                 gettext("%s: operand follows operand\n"),
447                                                 cmdname);
448                                 exit(1);
449                         }
450                         if (np->action != NOT && oldnp == 0)
451                                 goto err;
452                         wasop = argp->action;
453                 } else {
454                         wasop = PRINT;
455                         if (argp->type != Unary) {
456                                 if (!(b = *++av)) {
457                                         (void) fprintf(stderr,
458                                         gettext("%s: incomplete statement\n"),
459                                                         cmdname);
460                                         exit(1);
461                                 }
462                                 if (argp->type == Num) {
463                                         if (((argp->action == MAXDEPTH) ||
464                                             (argp->action == MINDEPTH)) &&
```



```
465                                             ((int)strtol(b, (char **)NULL,
466                                             10) < 0))
467                                                 errx(1,
468                                         gettext("%s: value must be positive"),
469                                                     (argp->action == MAXDEPTH) ?
470                                                     "maxdepth" : "mindepth");
471                                         if ((argp->action != PERM) ||
472                                             (*b != '+')) {
473                                                 if (*b == '+' || *b == '-') {
474                                                         np->second.i = *b;
475                                                         b++;
476                                                 }
477                                         }
478                                 }
479                         }
480                 }
481                 switch (argp->action) {
482                 case AND:
483                         break;
484                 case NOT:
485                         break;
486                 case OR:
487                         np->first.np = topnode;
488                         topnode = np;
489                         oldnp->next = 0;
490                         break;

492                 case LPAREN: {
493                         struct Node *save = topnode;
494                         topnode = np+1;
495                         paren_cnt++;
496                         i = compile(++av, topnode, actionp);
497                         np->first.np = topnode;
498                         topnode = save;
499                         av += i;
500                         oldnp = np;
501                         np += i + 1;
502                         oldnp->next = np;
503                         continue;
504                 }

506                 case RPAREN:
507                         if (paren_cnt <= 0) {
508                                 (void) fprintf(stderr,
509                                     gettext("%s: unmatched ')'\n"),
510                                     cmdname);
511                                 exit(1);
512                         }
513                         paren_cnt--;
514                         if (oldnp == 0)
515                                 goto err;
516                         if (oldnp->type == Op) {
517                                 (void) fprintf(stderr,
518                                     gettext("%s: cannot immediately"
519                                     " follow an operand with ')'\n"),
520                                     cmdname);
521                                 exit(1);
522                         }
523                         oldnp->next = 0;
524                         return (av-argv);

526                 case FOLLOW:
527                         walkflags &= ~FTW_PHYS;
528                         break;
529                 case MOUNT:
530                         walkflags |= FTW_MOUNT;
```

```
531                         break;
532                 case DEPTH:
533                         walkflags |= FTW_DEPTH;
534                         break;
535
536                 case LOCAL:
537                         np->first.l = 0L;
538                         np->first.ll = 0LL;
539                         np->second.i = '+';
540                         /*
541                          * Make it compatible to df -l for
542                          * future enhancement. So, anything
543                          * that is not remote, then it is
544                          * local.
545                          */
546                         init_remote_fs();
547                         break;
548
549                 case SIZE:
550                         if (b[strlen(b)-1] == 'c')
551                                 np->action = CSIZE;
552                         /*FALLTHROUGH*/
553                 case INUM:
554                         np->first.ll = atoll(b);
555                         break;
556
557                 case CMIN:
558                 case CTIME:
559                 case MMIN:
560                 case MTIME:
561                 case AMIN:
562                 case ATIME:
563                 case LINKS:
564                         np->first.l = atol(b);
565                         break;
566
567                 case F_USER:
568                 case F_GROUP:
569                 case F_USERACL:
570                 case F_GROUPACL: {
571                         struct  passwd  *pw;
572                         struct  group *gr;
573                         long value;
574                         char *q;
575
576                         value = -1;
577                         if (argp->action == F_USER ||
578                             argp->action == F_USERACL) {
579                                 if ((pw = getpwnam(b)) != 0)
580                                         value = (long)pw->pw_uid;
581                         } else {
582                                 if ((gr = getgrnam(b)) != 0)
583                                         value = (long)gr->gr_gid;
584                         }
585                         if (value == -1) {
586                                 errno = 0;
587                                 value = strtol(b, &q, 10);
588                                 if (errno != 0 || q == b || *q != '\0') {
589                                         (void) fprintf(stderr, gettext(
590                                             "%s: cannot find %s name\n"),
591                                                 cmdname, *av);
592                                         exit(1);
593                                 }
594                         }
595                         np->first.l = value;
596                         break;
```



```
597                 }
598
599                 case EXEC:
600                 case OK:
601                         walkflags &= ~FTW_CHDIR;
602                         np->first.ap = av;
603                         (*actionp)++;
604                         for (;;) {
605                                 if ((b = *av) == 0) {
606                                         (void) fprintf(stderr,
607                                         gettext("%s: incomplete statement\n"),
608                                                 cmdname);
609                                         exit(1);
610                                 }
611                                 if (strcmp(b, ";") == 0) {
612                                         *av = 0;
613                                         break;
614                                 } else if (strcmp(b, "{}") == 0)
615                                         *av = dummyarg;
616                                 else if (strcmp(b, "+") == 0 &&
617                                         av[-1] == dummyarg &&
618                                         np->action == EXEC) {
619                                         av[-1] = 0;
620                                         np->first.vp = varargs(np->first.ap);
621                                         np->action = VARARGS;
622                                         break;
623                                 }
624                                 av++;
625                         }
626                         break;
627
628                 case NAME:
629                 case INAME:
630                 case PATH:
631                 case IPATH:
632 #endif /* ! codereview */
633                         np->first.cp = b;
634                         break;
635                 case REGEX:
636                 case IREGEX: {
637                         int error;
638                         size_t errlen;
639                         char *errmsg;
640
641                         if ((preg = realloc(preg, (npreg + 1) *
642                             sizeof (regex_t))) == NULL)
643                                 err(1, "realloc");
644                         if ((error = regcomp(&preg[npreg], b,
645                             ((np->action == IREGEX) ? REG_ICASE : 0) |
646                             ((Eflag) ? REG_EXTENDED : 0))) != 0) {
647                                 errlen = regerror(error, &preg[npreg], NULL, 0);
648                                 if ((errmsg = malloc(errlen)) == NULL)
649                                         err(1, "malloc");
650                                 (void) regerror(error, &preg[npreg], errmsg,
651                                     errlen);
652                                 errx(1, gettext("RE error: %s"), errmsg);
653                         }
654                         npreg++;
655                         break;
656                 }
657                 case PERM:
658                         if (*b == '-')
659                                 ++b;
660
661                         if (readmode(b) != NULL) {
662                                 (void) fprintf(stderr, gettext(
```

```
663                                     "find: -perm: Bad permission string\n"));
664                             usage();
665                     }
666                     np->first.l = (long)getmode((mode_t)0);
667                     break;
668             case TYPE:
669                     i = *b;
670                     np->first.l =
671                         i == 'd' ? S_IFDIR :
672                         i == 'b' ? S_IFBLK :
673                         i == 'c' ? S_IFCHR :
674 #ifdef S_IFIFO
675                         i == 'p' ? S_IFIFO :
676 #endif
677                         i == 'f' ? S_IFREG :
678 #ifdef S_IFLNK
679                         i == 'l' ? S_IFLNK :
680 #endif
681 #ifdef S_IFSOCK
682                         i == 's' ? S_IFSOCK :
683 #endif
684 #ifdef S_IFDOOR
685                         i == 'D' ? S_IFDOOR :
686 #endif
687                         0;
688                     break;

690             case CPIO:
691                     if (walkflags & FTW_PHYS)
692                             com = cpio;
693                     else
694                             com = cpiol;
695                     goto common;

697             case NCPIO: {
698                     FILE *fd;

700                     if (walkflags & FTW_PHYS)
701                             com = ncpio;
702                     else
703                             com = ncpiol;
704             common:
705                     /* set up cpio */
706                     if ((fd = fopen(b, "w")) == NULL) {
707                             (void) fprintf(stderr,
708                                     gettext("%s: cannot create %s\n"),
709                                     cmdname, b);
710                             exit(1);
711                     }

713                     np->first.l = (long)cmdopen("cpio", com, "w", fd);
714                     (void) fclose(fd);
715                     walkflags |= FTW_DEPTH;
716                     np->action = CPIO;
717             }
718                     /*FALLTHROUGH*/
719             case PRINT:
720             case PRINT0:
721                     (*actionp)++;
722                     break;

724             case NEWER: {
725                     struct stat statb;
726                     if (stat(b, &statb) < 0) {
727                             (void) fprintf(stderr,
728                                     gettext("%s: cannot access %s\n"),
```



```
729                                     cmdname, b);
730                             exit(1);
731                     }
732                     np->first.l = statb.st_mtime;
733                     np->second.i = '+';
734                     break;
735             }

737             case PRUNE:
738             case NOUSER:
739             case NOGRP:
740                     break;
741             case FSTYPE:
742                     np->first.cp = b;
743                     break;
744             case LS:
745                     (*actionp)++;
746                     break;
747             case XATTR:
748                     break;
749             case ACL:
750                     break;
751             case MAXDEPTH:
752                     maxdepth = (int)strtol(b, (char **)NULL, 10);
753                     break;
754             case MINDEPTH:
755                     mindepth = (int)strtol(b, (char **)NULL, 10);
756                     break;
757             }

759             oldnp = np++;
760             oldnp->next = np;
761     }

763     if ((*av) || (wasop))
764             goto err;

766     if (paren_cnt != 0) {
767             (void) fprintf(stderr, gettext("%s: unmatched '('\n"),
768             cmdname);
769             exit(1);
770     }

772     /* just before returning, save next free node from the list */
773     freenode = oldnp->next;
774     oldnp->next = 0;
775     return (av-argv);
776 err:
777     if (*av)
778             (void) fprintf(stderr,
779                 gettext("%s: bad option %s\n"), cmdname, *av);
780     else
781             (void) fprintf(stderr, gettext("%s: bad option\n"), cmdname);
782     usage();
783     /*NOTREACHED*/
784 }

786 /*
787  * print out a usage message
788  */

790 static void
791 usage(void)
792 {
793     (void) fprintf(stderr,
794         gettext("%s: [-E] [-H | -L] path-list predicate-list\n"), cmdname);
```

```
795         exit(1);
796 }

798 /*
799  * This is the function that gets executed at each node
800  */

802 static int
803 execute(name, statb, type, state)
804 char *name;
805 struct stat *statb;
806 int type;
807 struct FTW *state;
808 {
809         struct Node *np = topnode;
810         int val;
811         time_t t;
812         long l;
813         long long ll;
814         int not = 1;
815         char *filename;
816         int cnpreg = 0;

818         if (type == FTW_NS) {
819                 (void) fprintf(stderr, gettext("%s: stat() error %s: %s\n"),
820                         cmdname, name, strerror(errno));
821                 error = 1;
822                 return (0);
823         } else if (type == FTW_DNR) {
824                 (void) fprintf(stderr, gettext("%s: cannot read dir %s: %s\n"),
825                         cmdname, name, strerror(errno));
826                 error = 1;
827         } else if (type == FTW_SLN && lflag == 1) {
828                 (void) fprintf(stderr,
829                         gettext("%s: cannot follow symbolic link %s: %s\n"),
830                         cmdname, name, strerror(errno));
831                 error = 1;
832         } else if (type == FTW_DL) {
833                 (void) fprintf(stderr, gettext("%s: cycle detected for %s\n"),
834                         cmdname, name);
835                 error = 1;
836                 return (0);
837         }

839         if ((maxdepth != -1 && state->level > maxdepth) ||
840             (mindepth != -1 && state->level < mindepth))
841                 return (0);

843         while (np) {
844                 switch (np->action) {
845                 case NOT:
846                         not = !not;
847                         np = np->next;
848                         continue;

850                 case AND:
851                         np = np->next;
852                         continue;

854                 case OR:
855                         if (np->first.np == np) {
856                                 /*
857                                  * handle naked OR (no term on left hand side)
858                                  */
859                                 (void) fprintf(stderr,
860                                     gettext("%s: invalid -o construction\n"),
```



```
861                                         cmdname);
862                                 exit(2);
863                         }
864                         /* FALLTHROUGH */
865                 case LPAREN: {
866                         struct Node *save = topnode;
867                         topnode = np->first.np;
868                         (void) execute(name, statb, type, state);
869                         val = lastval;
870                         topnode = save;
871                         if (np->action == OR) {
872                                 if (val)
873                                         return (0);
874                                 val = 1;
875                         }
876                         break;
877                 }

879                 case LOCAL: {
880                         int     nremfs;
881                         val = 1;
882                         /*
883                          * If file system type matches the remote
884                          * file system type, then it is not local.
885                          */
886                         for (nremfs = 0; nremfs < fstype_index; nremfs++) {
887                                 if (strcmp(remote_fstypes[nremfs],
888                                                 statb->st_fstype) == 0) {
889                                         val = 0;
890                                         break;
891                                 }
892                         }
893                         break;
894                 }

896                 case TYPE:
897                         l = (long)statb->st_mode&S_IFMT;
898                         goto num;

900                 case PERM:
901                         l = (long)statb->st_mode&07777;
902                         if (np->second.i == '-')
903                                 val = ((l&np->first.l) == np->first.l);
904                         else
905                                 val = (l == np->first.l);
906                         break;

908                 case INUM:
909                         ll = (long long)statb->st_ino;
910                         goto llnum;
911                 case NEWER:
912                         l = statb->st_mtime;
913                         goto num;
914                 case ATIME:
915                         t = statb->st_atime;
916                         goto days;
917                 case CTIME:
918                         t = statb->st_ctime;
919                         goto days;
920                 case MTIME:
921                         t = statb->st_mtime;
922                 days:
923                         l = (now-t)/A_DAY;
924                         goto num;
925                 case MMIN:
926                         t = statb->st_mtime;
```

```
927                     goto mins;
928             case AMIN:
929                     t = statb->st_atime;
930                     goto mins;
931             case CMIN:
932                     t = statb->st_ctime;
933                     goto mins;
934             mins:
935                     l = (now-t)/A_MIN;
936                     goto num;
937             case CSIZE:
938                     ll = (long long)statb->st_size;
939                     goto llnum;
940             case SIZE:
941                     ll = (long long)round(statb->st_size, BLKSIZ)/BLKSIZ;
942                     goto llnum;
943             case F_USER:
944                     l = (long)statb->st_uid;
945                     goto num;
946             case F_GROUP:
947                     l = (long)statb->st_gid;
948                     goto num;
949             case LINKS:
950                     l = (long)statb->st_nlink;
951                     goto num;
952             llnum:
953                     if (np->second.i == '+')
954                             val = (ll > np->first.ll);
955                     else if (np->second.i == '-')
956                             val = (ll < np->first.ll);
957                     else
958                             val = (ll == np->first.ll);
959                     break;
960             num:
961                     if (np->second.i == '+')
962                             val = (l > np->first.l);
963                     else if (np->second.i == '-')
964                             val = (l < np->first.l);
965                     else
966                             val = (l == np->first.l);
967                     break;
968             case OK:
969                     val = ok(name, np->first.ap);
970                     break;
971             case EXEC:
972                     val = doexec(name, np->first.ap, NULL);
973                     break;

975             case VARARGS: {
976                     struct Arglist *ap = np->first.vp;
977                     char *cp;
978                     cp = ap->nextstr - (strlen(name)+1);
979                     if (cp >= (char *)(ap->nextvar+3)) {
980                             /* there is room just copy the name */
981                             val = 1;
982                             (void) strcpy(cp, name);
983                             *ap->nextvar++ = cp;
984                             ap->nextstr = cp;
985                     } else {
986                             /* no more room, exec command */
987                             *ap->nextvar++ = name;
988                             *ap->nextvar = 0;
989                             val = 1;
990                             (void) doexec((char *)0, ap->arglist,
991                                 &exec_exitcode);
992                             ap->nextstr = ap->end;
```



```
993                             ap->nextvar = ap->firstvar;
994                     }
995                     break;
996             }

998             case DEPTH:
999             case MOUNT:
1000            case FOLLOW:
1001                    val = 1;
1002                    break;

1004            case NAME:
1005            case INAME:
1006            case PATH:
1007            case IPATH: {
1008                    char *path;
1009                    int fnmflags = 0;

1011                    if (np->action == INAME || np->action == IPATH)
1012                            fnmflags = FNM_IGNORECASE;
  66            case INAME: {
  67                    char *name1;
  68                    int fnmflags = (np->action == INAME) ?
  69                        FNM_IGNORECASE : 0;

1014                    /*
1015                     * basename(3c) may modify name, so
1016                     * we need to pass another string
1017                     */
1018                    if ((path = strdup(name)) == NULL) {
  75                    if ((name1 = strdup(name)) == NULL) {
1019                            (void) fprintf(stderr,
1020                                gettext("%s: cannot strdup() %s: %s\n"),
1021                                cmdname, name, strerror(errno));
1022                            exit(2);
1023                    }
1024                    /*
1025                     * XPG4 find should not treat a leading '.' in a
1026                     * filename specially for pattern matching.
1027                     * /usr/bin/find  will not pattern match a leading
1028                     * '.' in a filename, unless '.' is explicitly
1029                     * specified.
1030                     */
1031 #ifndef XPG4
1032                    fnmflags |= FNM_PERIOD;
1033 #endif

1035                    val = !fnmatch(np->first.cp,
1036                        (np->action == NAME || np->action == INAME)
1037                            ? basename(path) : path, fnmflags);
1038                    free(path);
  91                    val = !fnmatch(np->first.cp, basename(name1), fnmflags);
  92                    free(name1);
1039                    break;
1040            }

1042            case PRUNE:
1043                    if (type == FTW_D)
1044                            state->quit = FTW_PRUNE;
1045                    val = 1;
1046                    break;
1047            case NOUSER:
1048                    val = ((getpwuid(statb->st_uid)) == 0);
1049                    break;
1050            case NOGRP:
1051                    val = ((getgrgid(statb->st_gid)) == 0);
```

```
1052                    break;
1053            case FSTYPE:
1054                    val = (strcmp(np->first.cp, statb->st_fstype) == 0);
1055                    break;
1056            case CPIO:
1057                    output = (FILE *)np->first.l;
1058                    (void) fprintf(output, "%s\n", name);
1059                    val = 1;
1060                    break;
1061            case PRINT:
1062            case PRINT0:
1063                    (void) fprintf(stdout, "%s%c", name,
1064                        (np->action == PRINT) ? '\n' : '\0');
1065                    val = 1;
1066                    break;
1067            case LS:
1068                    (void) list(name, statb);
1069                    val = 1;
1070                    break;
1071            case XATTR:
1072                    filename = (walkflags & FTW_CHDIR) ?
1073                            gettail(name) : name;
1074                    val = (pathconf(filename, _PC_XATTR_EXISTS) == 1);
1075                    break;
1076            case ACL:
1077                    /*
1078                     * Need to get the tail of the file name, since we have
1079                     * already chdir()ed into the directory (performed in
1080                     * nftw()) of the file
1081                     */
1082                    filename = (walkflags & FTW_CHDIR) ?
1083                            gettail(name) : name;
1084                    val = acl_trivial(filename);
1085                    break;
1086            case F_USERACL:
1087            case F_GROUPACL: {
1088                    int i;
1089                    acl_t *acl;
1090                    void *acl_entry;
1091                    aclent_t *p1;
1092                    ace_t *p2;

1094                    filename = (walkflags & FTW_CHDIR) ?
1095                        gettail(name) : name;
1096                    val = 0;
1097                    if (acl_get(filename, 0, &acl) != 0)
1098                            break;
1099                    for (i = 0, acl_entry = acl->acl_aclp;
1100                        i != acl->acl_cnt; i++) {
1101                            if (acl->acl_type == ACLENT_T) {
1102                                    p1 = (aclent_t *)acl_entry;
1103                                    if (p1->a_id == np->first.l) {
1104                                            val = 1;
1105                                            acl_free(acl);
1106                                            break;
1107                                    }
1108                            } else {
1109                                    p2 = (ace_t *)acl_entry;
1110                                    if (p2->a_who == np->first.l) {
1111                                            val = 1;
1112                                            acl_free(acl);
1113                                            break;
1114                                    }
1115                            }
1116                            acl_entry = ((char *)acl_entry +
1117                                acl->acl_entry_size);
```



```
1118                    }
1119                    acl_free(acl);
1120                    break;
1121            }
1122            case IREGEX:
1123            case REGEX: {
1124                    regmatch_t pmatch;

1126                    val = 0;
1127                    if (regexec(&preg[cnpreg], name, 1, &pmatch, NULL) == 0)
1128                            val = ((pmatch.rm_so == 0) &&
1129                                (pmatch.rm_eo == strlen(name)));
1130                    cnpreg++;
1131                    break;
1132            }
1133            case MAXDEPTH:
1134                    if (state->level == maxdepth && type == FTW_D)
1135                            state->quit = FTW_PRUNE;
1136                    /* FALLTHROUGH */
1137            case MINDEPTH:
1138                    val = 1;
1139                    break;
1140            }
1141            /*
1142             * evaluate 'val' and 'not' (exclusive-or)
1143             * if no inversion (not == 1), return only when val == 0
1144             * (primary not true). Otherwise, invert the primary
1145             * and return when the primary is true.
1146             * 'Lastval' saves the last result (fail or pass) when
1147             * returning back to the calling routine.
1148             */
1149            if (val^not) {
1150                    lastval = 0;
1151                    return (0);
1152            }
1153            lastval = 1;
1154            not = 1;
1155            np = np->next;
1156        }
1157        return (0);
1158 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   24025 Wed May 29 12:55:55 2013
new/usr/src/man/man1/find.1
3795 find does not support -path or -ipath
**********************************************************
   1 '\" te
   2 .\" Copyright 1989 AT&T
   3 .\" Copyright (c) 2008, Sun Microsystems, Inc. All Rights Reserved
   4 .\" Copyright 2011 Nexenta Systems, Inc. All rights reserved.
   5 .\" Copyright (c) 2013 Andrew Stormont.  All rights reserved.
   6 #endif /* ! codereview */
   7 .\" Portions Copyright (c) 1992, X/Open Company Limited All Rights Reserved
   8 .\"
   9 .\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for
  10 .\" permission to reproduce portions of its copyrighted documentation.
  11 .\" Original documentation from The Open Group can be obtained online at
  12 .\" http://www.opengroup.org/bookstore/.
  13 .\" The Institute of Electrical and Electronics Engineers and The Open Group,
  14 .\" have given us permission to reprint portions of their documentation.
  15 .\" In the following statement, the phrase "this text" refers to portions of
  16 .\" the system documentation. Portions of this text are reprinted and
  17 .\" reproduced in electronic form in the Sun OS Reference Manual, from
  18 .\" IEEE Std 1003.1, 2004 Edition, Standard for Information Technology --
  19 .\" Portable Operating System Interface (POSIX), The Open Group Base
  20 .\" Specifications Issue 6, Copyright (C) 2001-2004 by the Institute of
  21 .\" Electrical and Electronics Engineers, Inc and The Open Group. In the event
  22 .\" of any discrepancy between these versions and the original IEEE and
  23 .\" The Open Group Standard, the original IEEE and The Open Group Standard
  24 .\" is the referee document. The original Standard can be obtained online
  25 .\" at http://www.opengroup.org/unix/online.html.
  26 .\" This notice shall appear on any product containing this material.
  27 .\" The contents of this file are subject to the terms of the Common Development
  28 .\" and Distribution License (the "License").  You may not use this file except
  29 .\" in compliance with the License.
  30 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or
  31 .\" http://www.opensolaris.org/os/licensing.  See the License for the specific
  32 .\" language governing permissions and limitations under the License.
  33 .\" When distributing Covered Code, include this CDDL HEADER in each file and
  34 .\" include the License file at usr/src/OPENSOLARIS.LICENSE.  If applicable,
  35 .\" add the following below this CDDL HEADER, with the fields enclosed by
  36 .\" brackets "[]" replaced with your own identifying information:
  37 .\" Portions Copyright [yyyy] [name of copyright owner]
  38 .TH FIND 1 "Sep 5, 2011"
  39 .SH NAME
  40 find \- find files
  41 .SH SYNOPSIS
  42 .LP
  43 .nf
  44 \fB/usr/bin/find\fR [\fB-E\fR] [\fB-H\fR | \fB-L\fR] \fIpath\fR... \fIexpression
  45 .fi

  47 .LP
  48 .nf
  49 \fB/usr/xpg4/bin/find\fR [\fB-H\fR | \fB-L\fR] \fIpath\fR... \fIexpression\fR
  50 .fi

  52 .SH DESCRIPTION
  53 .sp
  54 .LP
  55 The \fBfind\fR utility recursively descends the directory hierarchy for each
  56 \fIpath\fR seeking files that match a Boolean \fIexpression\fR written in the
  57 primaries specified below.
  58 .sp
  59 .LP
  60 \fBfind\fR is able to descend to arbitrary depths in a file hierarchy and does
  61 not fail due to path length limitations (unless a \fIpath\fR operand specified
```



```
  62 by the application exceeds \fIPATH_MAX\fR requirements).
  63 .sp
  64 .LP
  65 \fBfind\fR detects infinite loops; that is, entering a previously visited
  66 directory that is an ancestor of the last file encountered.
  67 .SH OPTIONS
  68 .sp
  69 .LP
  70 The following options are supported:
  71 .sp
  72 .ne 2
  73 .na
  74 \fB\fB-E\fR\fR
  75 .ad
  76 .RS 6n
  77 Interpret regular expressions followed by \fB-regex\fR and \fB-iregex\fR
  78 primaries as extended regular expressions.
  79 .RE

  81 .sp
  82 .ne 2
  83 .na
  84 \fB\fB-H\fR\fR
  85 .ad
  86 .RS 6n
  87 Causes the file information and file type evaluated for each symbolic link
  88 encountered on the command line to be those of the file referenced by the link,
  89 and not the link itself. If the referenced file does not exist, the file
  90 information and type is for the link itself. File information for all symbolic
  91 links not on the command line is that of the link itself.
  92 .RE

  94 .sp
  95 .ne 2
  96 .na
  97 \fB\fB-L\fR\fR
  98 .ad
  99 .RS 6n
 100 Causes the file information and file type evaluated for each symbolic link to
 101 be those of the file referenced by the link, and not the link itself. See
 102 \fBNOTES\fR.
 103 .RE

 105 .sp
 106 .LP
 107 Specifying more than one of the mutually-exclusive options \fB-H\fR and
 108 \fB-L\fR is not considered an error. The last option specified determines the
 109 behavior of the utility.
 110 .SH OPERANDS
 111 .sp
 112 .LP
 113 The following operands are supported:
 114 .sp
 115 .ne 2
 116 .na
 117 \fB\fIpath\fR\fR
 118 .ad
 119 .RS 14n
 120 A pathname of a starting point in the directory hierarchy.
 121 .RE

 123 .sp
 124 .ne 2
 125 .na
 126 \fB\fIexpression\fR\fR
 127 .ad
```

```
.RS 14n
The first argument that starts with a \fB\(mi\fR, or is a \fB!\fR or a \fB(\fR,
and all subsequent arguments are interpreted as an \fIexpression\fR made up of
the following primaries and operators. In the descriptions, wherever \fIn\fR is
used as a primary argument, it is interpreted as a decimal integer optionally
preceded by a plus (\fB+\fR) or minus (\fB\(mi\fR) sign, as follows:
.sp
.ne 2
.na
\fB+\fIn\fR\fR
.ad
.RS 6n
more than \fIn\fR
.RE

.sp
.ne 2
.na
\fB\fIn\fR\fR
.ad
.RS 6n
exactly \fIn\fR
.RE

.sp
.ne 2
.na
\fB-\fIn\fR\fR
.ad
.RS 6n
less than \fIn\fR
.RE

.RE

.SS "Expressions"
.sp
.LP
Valid expressions are:
.sp
.ne 2
.na
\fB\fB-acl\fR\fR
.ad
.RS 17n
True if the file have additional ACLs defined.
.RE

.sp
.ne 2
.na
\fB\fB-amin\fR \fIn\fR\fR
.ad
.RS 17n
File was last accessed \fIn\fR minutes ago.
.RE

.sp
.ne 2
.na
\fB\fB-atime\fR \fIn\fR\fR
.ad
.RS 17n
True if the file was accessed \fIn\fR days ago. The access time of directories
in \fIpath\fR is changed by \fBfind\fR itself.
.RE
```



```
.sp
.ne 2
.na
\fB\fB-cmin\fR \fIn\fR\fR
.ad
.RS 17n
File's status was last changed \fIn\fR minutes ago.
.RE

.sp
.ne 2
.na
\fB\fB-cpio\fR \fIdevice\fR\fR
.ad
.RS 17n
Always true. Writes the current file on \fIdevice\fR in \fBcpio\fR format
(5120-byte records).
.RE

.sp
.ne 2
.na
\fB\fB-ctime\fR \fIn\fR\fR
.ad
.RS 17n
True if the file's status was changed \fIn\fR days ago.
.RE

.sp
.ne 2
.na
\fB\fB-depth\fR\fR
.ad
.RS 17n
Always true. Causes descent of the directory hierarchy to be done so that all
entries in a directory are acted on before the directory itself. This can be
useful when \fBfind\fR is used with \fBcpio\fR(1) to transfer files that are
contained in directories without write permission.
.RE

.sp
.ne 2
.na
\fB\fB-exec\fR \fIcommand\fR\fR
.ad
.RS 17n
True if the executed command returns a zero value as exit status. The end of
command must be punctuated by an escaped semicolon (\fB;\fR). A command
argument \fB{}\fR is replaced by the current pathname. If the last argument to
\fB-exec\fR is \fB{}\fR and you specify \fB+\fR rather than the semicolon
(\fB;\fR), the command is invoked fewer times, with \fB{}\fR replaced by groups
of pathnames. If any invocation of the command returns a non-zero value as exit
status, find returns a non-zero exit status.
.RE

.sp
.ne 2
.na
\fB\fB-follow\fR\fR
.ad
.RS 17n
Always true and always evaluated no matter where it appears in
\fIexpression\fR. The behavior is unspecified if \fB-follow\fR is used when the
\fBfind\fR command is invoked with either the \fB-H\fR or the \fB-L\fR option.
Causes symbolic links to be followed. When following symbolic links, \fBfind\fR
```

```
keeps track of the directories visited so that it can detect infinite loops.
For example, such a loop would occur if a symbolic link pointed to an ancestor.
This expression should not be used with the find-type \fBl\fR expression. See
\fBNOTES\fR.
.RE

.sp
.ne 2
.na
\fB\fB-fstype\fR \fItype\fR\fR
.ad
.RS 17n
True if the filesystem to which the file belongs is of type \fItype\fR.
.RE

.sp
.ne 2
.na
\fB\fB-group\fR \fIgname\fR\fR
.ad
.RS 17n
True if the file belongs to the group \fIgname\fR. If \fIgname\fR is numeric
and there's no such group name, it is taken as a group \fBID\fR.
.RE

.sp
.ne 2
.na
\fB\fB-groupacl\fR \fIgname\fR\fR
.ad
.RS 17n
True if the file's ACL contains an entry for the group \fIgname\fR.
If \fIgname\fR is numeric and there's no such group name, it is taken
as a group \fBID\fR.
.RE

.sp
.ne 2
.na
\fB\fB-iname\fR \fIpattern\fR\fR
.ad
.RS 17n
Like \fB-name\fR, but the match is case insensitive.
.RE

.sp
.ne 2
.na
\fB\fB-inum\fR \fIn\fR\fR
.ad
.RS 17n
True if the file has inode number \fIn\fR.
.RE

.sp
.ne 2
.na
\fB\fB-ipath\fR \fIpattern\fR\fR
.ad
.RS 17n
Like \fB-path\fR, but the match is case insensitive.
.RE

.sp
.ne 2
.na
```



```
#endif /* ! codereview */
\fB\fB-iregex\fR \fIpattern\fR\fR
.ad
.RS 17n
Like \fB-regex\fR, but the match is case insensitive.
.RE

.sp
.ne 2
.na
\fB\fB-links\fR \fIn\fR\fR
.ad
.RS 17n
True if the file has \fIn\fR links.
.RE

.sp
.ne 2
.na
\fB\fB-local\fR\fR
.ad
.RS 17n
True if the file system type is not a remote file system type as defined in the
\fB/etc/dfs/fstypes\fR file. \fBnfs\fR is used as the default remote filesystem
type if the \fB/etc/dfs/fstypes\fR file is not present. The \fB-local\fR option
descends the hierarchy of non-local directories. See \fBEXAMPLES\fR for an
example of how to search for local files without descending.
.RE

.sp
.ne 2
.na
\fB\fB-ls\fR\fR
.ad
.RS 17n
Always true. Prints current pathname together with its associated statistics.
These include (respectively):
.RS +4
.TP
.ie t \(bu
.el o
inode number
.RE
.RS +4
.TP
.ie t \(bu
.el o
size in kilobytes (1024 bytes)
.RE
.RS +4
.TP
.ie t \(bu
.el o
protection mode
.RE
.RS +4
.TP
.ie t \(bu
.el o
number of hard links
.RE
.RS +4
.TP
.ie t \(bu
.el o
user
```

```
.RE
.RS +4
.TP
.ie t \(bu
.el o
group
.RE
.RS +4
.TP
.ie t \(bu
.el o
size in bytes
.RE
.RS +4
.TP
.ie t \(bu
.el o
modification time.
.RE
If the file is a special file, the size field instead contains the major and
minor device numbers.
.sp
If the file is a symbolic link, the pathname of the linked-to file is printed
preceded by ‘\fB\(->\fR\&’. The format is identical to that of \fBls\fR
\fB-gilds\fR (see \fBls\fR(1B)).
.sp
Formatting is done internally, without executing the \fBls\fR program.
.RE

.sp
.ne 2
.na
\fB\fB-maxdepth\fR \fIn\fR\fR
.ad
.RS 17n
Always true; descend at most \fIn\fR directory levels below the command
line arguments. If any \fB-maxdepth\fR primary is specified, it
applies to the entire expression even if it would not normally be
evaluated. \fB-maxdepth 0\fR limits the whole search to
the command line arguments.
.RE

.sp
.ne 2
.na
\fB\fB-mindepth\fR \fIn\fR\fR
.ad
.RS 17n
Always true; do not apply any tests or actions at levels less
than \fIn\fR. If any \fB-mindepth\fR primary is specified, it applies to the
entire expression even if it would not normally be evaluated.
\fB-mindepth 1\fR processes all but the command line arguments.
.RE

.sp
.ne 2
.na
\fB\fB-mmin\fR \fIn\fR\fR
.ad
.RS 17n
File’s data was last modified \fIn\fR minutes ago.
.RE

.sp
.ne 2
.na
```



```
\fB\fB-mount\fR\fR
.ad
.RS 17n
Always true. Restricts the search to the file system containing the directory
specified. Does not list mount points to other file systems.
.RE

.sp
.ne 2
.na
\fB\fB-mtime\fR \fIn\fR\fR
.ad
.RS 17n
True if the file’s data was modified \fIn\fR days ago.
.RE

.sp
.ne 2
.na
\fB\fB-name\fR \fIpattern\fR\fR
.ad
.RS 17n
True if \fIpattern\fR matches the basename of the current file name. Normal
shell file name generation characters (see \fBsh\fR(1)) can be used. A
backslash (\fB\|\e\|\fR) is used as an escape character within the pattern. The
pattern should be escaped or quoted when \fBfind\fR is invoked from the shell.
.sp
Unless the character ’\fB\&.\fR’ is explicitly specified in the beginning of
\fIpattern\fR, a current file name beginning with ’\fB\&.\fR’ does not match
\fIpattern\fR when using \fB/usr/bin/find\fR. \fB/usr/xpg4/bin/find\fR does not
make this distinction; wildcard file name generation characters can match file
names beginning with ’\fB\&.\fR’.
.RE

.sp
.ne 2
.na
\fB\fB-ncpio\fR \fIdevice\fR\fR
.ad
.RS 17n
Always true. Writes the current file on \fIdevice\fR in \fBcpio\fR \fB-c\fR
format (5120 byte records).
.RE

.sp
.ne 2
.na
\fB\fB-newer\fR \fIfile\fR\fR
.ad
.RS 17n
True if the current file has been modified more recently than the argument
\fIfile\fR.
.RE

.sp
.ne 2
.na
\fB\fB-nogroup\fR\fR
.ad
.RS 17n
True if the file belongs to non-existing group.
.RE

.sp
.ne 2
.na
```

```
\fB\fB-nouser\fR\fR
.ad
.RS 17n
True if the file belongs to non-existing user.
.RE

.sp
.ne 2
.na
\fB\fB-ok\fR \fIcommand\fR\fR
.ad
.RS 17n
Like \fB-exec\fR, except that the generated command line is printed with a
question mark first, and is executed only if the response is affirmative.
.RE

.sp
.ne 2
.na
\fB\fB-path\fR\fR
.ad
.RS 17n
Like \fB-name\fR, but matches the entire file path and not just basename.
.RE

.sp
.ne 2
.na
#endif /* ! codereview */
\fB\fB-perm\fR [\fB-\fR]\fImode\fR\fR
.ad
.RS 17n
The \fImode\fR argument is used to represent file mode bits. It is identical in
format to the symbolic mode operand, \fIsymbolic_mode_list\fR, described in
\fBchmod\fR(1), and is interpreted as follows. To start, a template is assumed
with all file mode bits cleared. An \fIop\fR symbol of:
.sp
.ne 2
.na
\fB\fB+\fR\fR
.ad
.RS 8n
Set the appropriate mode bits in the template
.RE

.sp
.ne 2
.na
\fB\fB\(mi\fR\fR
.ad
.RS 8n
Clear the appropriate bits
.RE

.sp
.ne 2
.na
\fB\fB=\fR\fR
.ad
.RS 8n
Set the appropriate mode bits, without regard to the contents of the file mode
creation mask of the process
.RE

The \fIop\fR symbol of \fB\(mi\fR cannot be the first character of \fImode\fR,
to avoid ambiguity with the optional leading hyphen. Since the initial mode is
```



```
all bits off, there are no symbolic modes that need to use \fB\(mi\fR as the
first character.
.sp
If the hyphen is omitted, the primary evaluates as true when the file
permission bits exactly match the value of the resulting template.
.sp
Otherwise, if \fImode\fR is prefixed by a hyphen, the primary evaluates as true
if at least all the bits in the resulting template are set in the file
permission bits.
.RE

.sp
.ne 2
.na
\fB\fB-perm\fR [\fB-\fR]\fIonum\fR\fR
.ad
.RS 17n
True if the file permission flags exactly match the octal number \fIonum\fR
(see \fBchmod\fR(1)). If \fIonum\fR is prefixed by a minus sign (\fB\(mi\fR),
only the bits that are set in \fIonum\fR are compared with the file permission
flags, and the expression evaluates true if they match.
.RE

.sp
.ne 2
.na
\fB\fB-print\fR\fR
.ad
.RS 17n
Always true. Causes the current pathname to be printed.
.RE

.sp
.ne 2
.na
\fB\fB-print0\fR\fR
.ad
.RS 17n
Always true. Causes the current pathname to be printed, terminated by an ASCII
NUL character (character code 0) instead of a newline.
.RE

.sp
.ne 2
.na
\fB\fB-prune\fR\fR
.ad
.RS 17n
Always yields true. Does not examine any directories or files in the directory
structure below the \fIpattern\fR just matched. (See EXAMPLES). If \fB-depth\fR
is specified, \fB-prune\fR has no effect.
.RE

.sp
.ne 2
.na
\fB\fB-regex\fR \fIpattern\fR\fB
.ad
.RS 17n
True if the full path of the file matches \fIpattern\fR using regular
expressions.
.RE

.sp
.ne 2
.na
```

```
\fB\fB-size\fR \fIn\fR[\fBc\fR]\fR
.ad
.RS 17n
True if the file is \fIn\fR blocks long (512 bytes per block). If \fIn\fR is
followed by a \fBc\fR, the size is in bytes.
.RE

.sp
.ne 2
.na
\fB\fB-type\fR \fIc\fR\fR
.ad
.RS 17n
True if the type of the file is \fIc\fR, where \fIc\fR is \fBb\fR, \fBc\fR,
\fBd\fR, \fBD\fR, \fBf\fR, \fBl\fR, \fBp\fR, or \fBs\fR for block special file,
character special file, directory, door, plain file, symbolic link, fifo (named
pipe), or socket, respectively.
.RE

.sp
.ne 2
.na
\fB\fB-user\fR \fIuname\fR\fR
.ad
.RS 17n
True if the file belongs to the user \fIuname\fR. If \fIuname\fR is numeric and
there's no such user name, it is taken as a user \fBID\fR.
.RE

.sp
.ne 2
.na
\fB\fB-useracl\fR \fIuname\fR\fR
.ad
.RS 17n
True if the file's ACL contains an entry for the user \fIuname\fR.
If \fIuname\fR is numeric and there's no such user name, it is
taken as a user \fBID\fR.
.RE

.sp
.ne 2
.na
\fB\fB-xdev\fR\fR
.ad
.RS 17n
Same as the \fB-mount\fR primary.
.RE

.sp
.ne 2
.na
\fB\fB-xattr\fR\fR
.ad
.RS 17n
True if the file has extended attributes.
.RE

.SS "Complex Expressions"
.sp
.LP
The primaries can be combined using the following operators (in order of
decreasing precedence):
.sp
.ne 2
.na
```



```
\fB1)\fB(\fR\fIexpression\fR\fB)\fR\fR
.ad
.sp .6
.RS 4n
True if the parenthesized expression is true (parentheses are special to the
shell and must be escaped).
.RE

.sp
.ne 2
.na
\fB2)\fB!\fR\fIexpression\fR\fR
.ad
.sp .6
.RS 4n
The negation of a primary (\fB!\fR is the unary \fInot\fR operator).
.RE

.sp
.ne 2
.na
\fB3) \fIexpression\fR\fB[\fR\fB-a\fR\fB]\fR \fIexpression\fR\fR
.ad
.sp .6
.RS 4n
Concatenation of primaries (the \fIand\fR operation is implied by the
juxtaposition of two primaries).
.RE

.sp
.ne 2
.na
\fB4) \fIexpression\fR\fB\fR\fB-o\fR\fIexpression\fR\fR
.ad
.sp .6
.RS 4n
Alternation of primaries (\fB-o\fR is the \fIor\fR operator).
.RE

.sp
.LP
When you use \fBfind\fR in conjunction with \fBcpio\fR, if you use the \fB-L\fR
option with \fBcpio\fR, you must use the \fB-L\fR option or the \fB-follow\fR
primitive with \fBfind\fR and vice versa. Otherwise the results are
unspecified.
.sp
.LP
If no \fIexpression\fR is present, \fB-print\fR is used as the expression.
Otherwise, if the specified expression does not contain any of the primaries
\fB-exec\fR, \fB-ok\fR, \fB-ls\fR, or \fB-print\fR, the specified expression is
effectively replaced by:
.sp
.LP
(\fIspecified\fR) \fB-print\fR
.sp
.LP
The \fB-user\fR, \fB-group\fR, and \fB-newer\fR primaries each evaluate their
respective arguments only once. Invocation of \fIcommand\fR specified by
\fB-exec\fR or \fB-ok\fR does not affect subsequent primaries on the same file.
.SH USAGE
.sp
.LP
See \fBlargefile\fR(5) for the description of the behavior of \fBfind\fR when
encountering files greater than or equal to 2 Gbyte (2^31 bytes).
.SH EXAMPLES
.LP
```

```
\fBExample 1 \fRWriting Out the Hierarchy Directory
.sp
.LP
The following commands are equivalent:

.sp
.in +2
.nf
example% \fBfind .\fR
example% \fBfind . -print\fR
.fi
.in -2
.sp

.sp
.LP
They both write out the entire directory hierarchy from the current directory.

.LP
\fBExample 2 \fRRemoving Files
.sp
.LP
The following comand removes all files in your home directory named \fBa.out\fR
or \fB*.o\fR that have not been accessed for a week:

.sp
.in +2
.nf
example% \fBfind $HOME \e( -name a.out -o -name '*.o' \e) \e
       -atime +7 -exec rm {} \e;\fR
.fi
.in -2
.sp

.LP
\fBExample 3 \fRPrinting All File Names But Skipping SCCS Directories
.sp
.LP
The following command recursively print all file names in the current directory
and below, but skipping \fBSCCS\fR directories:

.sp
.in +2
.nf
example% \fBfind . -name SCCS -prune -o -print\fR
.fi
.in -2
.sp

.LP
\fBExample 4 \fRPrinting all file names and the SCCS directory name
.sp
.LP
Recursively print all file names in the current directory and below, skipping
the contents of \fBSCCS\fR directories, but printing out the \fBSCCS\fR
directory name:

.sp
.in +2
.nf
example% \fBfind . -print -name SCCS -prune\fR
.fi
.in -2
.sp

.LP
```



```
\fBExample 5 \fRTesting for the Newer File
.sp
.LP
The following command is basically equivalent to the \fB-nt\fR extension to
\fBtest\fR(1):

.sp
.in +2
.nf
example$ \fBif [ -n "$(find
file1 -prune -newer file2)" ]; then

printf %s\e\en "file1 is newer than file2"\fR
.fi
.in -2
.sp

.LP
\fBExample 6 \fRSelecting a File Using 24-hour Mode
.sp
.LP
The descriptions of \fB-atime\fR, \fB-ctime\fR, and \fB-mtime\fR use the
terminology \fIn\fR ''24-hour periods''. For example, a file accessed at 23:59
is selected by:

.sp
.in +2
.nf
example% \fBfind . -atime -1 -print\fR
.fi
.in -2
.sp

.sp
.LP
at 00:01 the next day (less than 24 hours later, not more than one day ago).
The midnight boundary between days has no effect on the 24-hour calculation.

.LP
\fBExample 7 \fRPrinting Files Matching a User's Permission Mode
.sp
.LP
The following command recursively print all file names whose permission mode
exactly matches read, write, and execute access for user, and read and execute
access for group and other:

.sp
.in +2
.nf
example% \fBfind . -perm u=rwx,g=rx,o=rx\fR
.fi
.in -2
.sp

.sp
.LP
The above could alternatively be specified as follows:

.sp
.in +2
.nf
example% \fBfind . -perm a=rwx,g-w,o-w\fR
.fi
.in -2
.sp
```

```
.LP
\fBExample 8 \fRPrinting Files with Write Access for \fBother\fR
.sp
.LP
The following command recursively print all file names whose permission
includes, but is not limited to, write access for other:

.sp
.in +2
.nf
example% \fBfind . -perm -o+w\fR
.fi
.in -2
.sp

.LP
\fBExample 9 \fRPrinting Local Files without Descending Non-local Directories
.sp
.in +2
.nf
example% \fBfind . ! -local -prune -o -print\fR
.fi
.in -2
.sp

.LP
\fBExample 10 \fRPrinting the Files in the Name Space Possessing Extended
Attributes
.sp
.in +2
.nf
example% \fBfind . -xattr\fR
.fi
.in -2
.sp

.SH ENVIRONMENT VARIABLES
.sp
.LP
See \fBenviron\fR(5) for descriptions of the following environment variables
that affect the execution of \fBfind\fR: \fBLANG\fR, \fBLC_ALL\fR,
\fBLC_COLLATE\fR, \fBLC_CTYPE\fR, \fBLC_MESSAGES\fR, and \fBNLSPATH\fR.
.sp
.ne 2
.na
\fB\fBPATH\fR\fR
.ad
.RS 8n
Determine the location of the \fIutility_name\fR for the \fB-exec\fR and
\fB-ok\fR primaries.
.RE

.sp
.LP
Affirmative responses are processed using the extended regular expression
defined for the \fByesexpr\fR keyword in the \fBLC_MESSAGES\fR category of the
user's locale. The locale specified in the \fBLC_COLLATE\fR category defines
the behavior of ranges, equivalence classes, and multi-character collating
elements used in the expression defined for \fByesexpr\fR. The locale specified
in \fBLC_CTYPE\fR determines the locale for interpretation of sequences of
bytes of text data a characters, the behavior of character classes used in the
expression defined for the \fByesexpr\fR. See \fBlocale\fR(5).
.SH EXIT STATUS
.sp
.LP
The following exit values are returned:
```



```
.sp
.ne 2
.na
\fB\fB0\fR\fR
.ad
.RS 6n
All \fIpath\fR operands were traversed successfully.
.RE

.sp
.ne 2
.na
\fB\fB>0\fR\fR
.ad
.RS 6n
An error occurred.
.RE

.SH FILES
.sp
.ne 2
.na
\fB\fB/etc/passwd\fR\fR
.ad
.RS 20n
Password file
.RE

.sp
.ne 2
.na
\fB\fB/etc/group\fR\fR
.ad
.RS 20n
Group file
.RE

.sp
.ne 2
.na
\fB\fB/etc/dfs/fstypes\fR\fR
.ad
.RS 20n
File that registers distributed file system packages
.RE

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
CSI     Enabled
_
Interface Stability     Committed
_
Standard        See \fBstandards\fR(5).
.TE
```

```
.SH SEE ALSO
.sp
.LP
\fBchmod\fR(1), \fBcpio\fR(1), \fBsh\fR(1), \fBtest\fR(1), \fBls\fR(1B),
\fBacl\fR(5), \fBregex\fR(5), \fBstat\fR(2), \fBumask\fR(2),
\fBattributes\fR(5), \fBenviron\fR(5), \fBfsattr\fR(5), \fBlargefile\fR(5),
\fBlocale\fR(5), \fBstandards\fR(5)
.SH WARNINGS
.sp
.LP
The following options are obsolete and will not be supported in future
releases:
.sp
.ne 2
.na
\fB\fB-cpio\fR \fIdevice\fR\fR
.ad
.RS 17n
Always true. Writes the current file on \fIdevice\fR in \fBcpio\fR format
(5120-byte records).
.RE

.sp
.ne 2
.na
\fB\fB-ncpio\fR \fIdevice\fR\fR
.ad
.RS 17n
Always true. Writes the current file on \fIdevice\fR in \fBcpio\fR \fB-c\fR
format (5120-byte records).
.RE

.SH NOTES
.sp
.LP
When using \fBfind\fR to determine files modified within a range of time, use
the \fB-mtime\fR argument \fBbefore\fR the \fB-print\fR argument. Otherwise,
\fBfind\fR gives all files.
.sp
.LP
Some files that might be under the Solaris root file system are actually mount
points for virtual file systems, such as \fBmntfs\fR or \fBnamefs\fR. When
comparing against a \fBufs\fR file system, such files are not selected if
\fB-mount\fR or \fB-xdev\fR is specified in the \fBfind\fR expression.
.sp
.LP
Using the \fB-L\fR or \fB-follow\fR option is not recommended when descending a
file-system hierarchy that is under the control of other users. In particular,
when using \fB-exec\fR, symbolic links can lead the \fBfind\fR command out of
the hierarchy in which it started. Using \fB-type\fR is not sufficient to
restrict the type of files on which the \fB-exec\fR command operates, because
there is an inherent race condition between the type-check performed by the
\fBfind\fR command and the time the executed command operates on the file
argument.
```