

```

*****
10185 Mon Sep 17 14:32:32 2012
new/usr/src/cmd/grep/grep.c
3047 grep support for -r would be useful
*****
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 2005 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  */
27 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
28 /*      All Rights Reserved */
30 /*      Copyright (c) 1987, 1988 Microsoft Corporation */
31 /*      All Rights Reserved */
33 /* Copyright 2012 Nexenta Systems, Inc. All rights reserved. */
35 /*
36  * grep -- print lines matching (or not matching) a pattern
37  *
38  *      status returns:
39  *          0 - ok, and some matches
40  *          1 - ok, but no matches
41  *          2 - some error
42  */
44 #include <sys/types.h>
46 #include <ctype.h>
47 #include <fcntl.h>
48 #include <locale.h>
49 #include <memory.h>
50 #include <regex.h>
51 #include <stdio.h>
52 #include <stdlib.h>
53 #include <string.h>
54 #include <unistd.h>
55 #include <ftw.h>
56 #include <limits.h>
57 #include <sys/param.h>
59 static const char *errstr[] = {
60     "Range endpoint too large.",
61     "Bad number.",

```

```

62     "'\\digit' out of range.",
63     "No remembered search string.",
64     "\\( \\) imbalance.",
65     "Too many \\(.",
66     "More than 2 numbers given in \\{ \\}.",
67     "} expected after \\.",
68     "First number exceeds second in \\{ \\}.",
69     "[ ] imbalance.",
70     "Regular expression overflow.",
71     "Illegal byte sequence.",
72     "Unknown regexp error code!!",
73     NULL
74 };
76 #define errmsg(msg, arg)      (void) fprintf(stderr, gettext(msg), arg)
77 #define BLKSIZE 512
78 #define GBUFSIZ 8192
79 #define MAX_DEPTH 1000
81 static int      temp;
82 static long long lnum;
83 static char     *linebuf;
84 static char     *prntbuf = NULL;
85 static long     fw_lPrntBufLen = 0;
86 static int      nflag;
87 static int      bflag;
88 static int      lflag;
89 static int      cflag;
90 static int      rflag;
91 static int      Rflag;
92 static int      vflag;
93 static int      sflag;
94 static int      iflag;
95 static int      wflag;
96 static int      hflag;
97 static int      qflag;
98 static int      errflag;
99 static int      nfile;
100 static long long tln;
101 static int      nsucc;
102 static int      outfn = 0;
103 static int      nlf;
104 static char     *ptr, *ptrend;
105 static char     *expbuf;
107 static void     execute(const char *, int);
108 static void     execute(char *);
109 static void     regerr(int);
110 static void     prepare(const char *);
111 static int      recursive(const char *, const struct stat *, int, struct FTW *);
112 static int      succeed(const char *);
113 static int      succeed(char *);
114 int
115 main(int argc, char **argv)
116 {
117     int      c;
118     char     *arg;
119     extern int optind;
120     (void) setlocale(LC_ALL, "");
121     #if !defined(TEXT_DOMAIN)
122     /* Should be defined by cc -D */
123     #define TEXT_DOMAIN "SYS_TEST" /* Use this only if it weren't */
124     #endif
125     (void) textdomain(TEXT_DOMAIN);

```

```

126 while ((c = getopt(argc, argv, "hqbclnRrsviyw")) != -1)
127 while ((c = getopt(argc, argv, "hqbclnRrsviyw")) != -1)
128     switch (c) {
129     case 'h':
130         hflag++;
131         break;
132     case 'q': /* POSIX: quiet: status only */
133         qflag++;
134         break;
135     case 'v':
136         vflag++;
137         break;
138     case 'c':
139         cflag++;
140         break;
141     case 'n':
142         nflag++;
143         break;
144     case 'R':
145         Rflag++;
146         /* FALLTHROUGH */
147     case 'r':
148         rflag++;
149         break;
150     case 'b':
151         bflag++;
152         break;
153     case 's':
154         sflag++;
155         break;
156     case 'l':
157         lflag++;
158         break;
159     case 'y':
160     case 'i':
161         iflag++;
162         break;
163     case 'w':
164         wflag++;
165         break;
166     case '?':
167         errflg++;
168     }
169
170 if (errflg || (optind >= argc)) {
171     errmsg("Usage: grep [-c|-l|-q] [-r|-R] -hbnsviw "
172           "pattern file . . .\n",
173           errmsg("Usage: grep [-c|-l|-q] -hbnsviw pattern file . . .\n",
174                 (char *)NULL));
175     exit(2);
176 }
177
178 argv = &argv[optind];
179 argc -= optind;
180 nfile = argc - 1;
181
182 if (strrchr(*argv, '\n') != NULL)
183     regerr(41);
184
185 if (iflag) {
186     for (arg = *argv; *arg != NULL; ++arg)
187         *arg = (char)tolower((int)((unsigned char)*arg));
188 }
189
190 if (wflag) {
191     unsigned int    wordlen;

```

```

190     char            *wordbuf;
191
192     wordlen = strlen(*argv) + 5; /* '\\\ ' <' *argv '\\\ ' >' '\0' */
193     if ((wordbuf = malloc(wordlen)) == NULL) {
194         errmsg("grep: Out of memory for word\n", (char *)NULL);
195         exit(2);
196     }
197
198     (void) strcpy(wordbuf, "\\<");
199     (void) strcat(wordbuf, *argv);
200     (void) strcat(wordbuf, "\\>");
201     *argv = wordbuf;
202 }
203
204 expbuf = compile(*argv, (char *)0, (char *)0);
205 if (regerrno)
206     regerr(regerrno);
207
208 if (--argc == 0)
209     execute(NULL, 0);
210 else
211     while (argc-- > 0)
212         prepare(++argv);
213         execute(++argv);
214
215 return (nsucc == 2 ? 2 : (nsucc == 0 ? 1 : 0));
216 }
217
218 static void
219 prepare(const char *path)
220 execute(char *file)
221 {
222     struct stat st;
223     int walkflags = FTW_CHDIR;
224     char *buf = NULL;
225
226     if (rflag) {
227         if (stat(path, &st) != -1 &&
228             (st.st_mode & S_IFMT) == S_IFDIR) {
229             outfn = 1;
230
231             /*
232              * Add trailing slash if arg
233              * is directory, to resolve symlinks.
234              */
235             if (path[strlen(path) - 1] != '/') {
236                 (void) asprintf(&buf, "%s/", path);
237                 if (buf != NULL)
238                     path = buf;
239             }
240
241             /*
242              * Search through subdirs if path is directory.
243              * Don't follow symlinks if Rflag is not set.
244              */
245             if (!Rflag)
246                 walkflags |= FTW_PHYS;
247
248             if (nftw(path, recursive, MAX_DEPTH, walkflags) != 0) {
249                 if (!sflag)
250                     errmsg("grep: can't open %s\n", path);
251                 nsucc = 2;
252             }
253         }
254     }
255     return;
256 }

```

```

253     }
254     execute(path, 0);
255 }

257 static int
258 recursive(const char *name, const struct stat *statp, int info, struct FTW *ftw)
259 {
260     /*
261      * process files and follow symlinks if Rflag set.
262      */
263     if (info != FTW_F) {
264         if (!sflag &&
265             (info == FTW_SLN || info == FTW_DNR || info == FTW_NS)) {
266             /* report broken symlinks and unreadable files */
267             errmsg("grep: can't open %s\n", name);
268         }
269         return (0);
270     }

271     /* skip devices and pipes if Rflag is not set */
272     if (!Rflag && !S_ISREG(statp->st_mode))
273         return (0);

274     /* pass offset to relative name from FTW_CHDIR */
275     execute(name, ftw->base);
276     return (0);
277 }

281 static void
282 execute(const char *file, int base)
283 {
284     char    *lbuf, *p;
285     long    count;
286     long    offset = 0;
287     char    *next_ptr = NULL;
288     long    next_count = 0;

290     tln = 0;

292     if (prntbuf == NULL) {
293         fw_lPrntBufLen = GBUFSIZ + 1;
294         if ((prntbuf = malloc(fw_lPrntBufLen)) == NULL) {
295             exit(2); /* out of memory - BAIL */
296         }
297         if ((linebuf = malloc(fw_lPrntBufLen)) == NULL) {
298             exit(2); /* out of memory - BAIL */
299         }
300     }

302     if (file == NULL)
303         temp = 0;
304     else if ((temp = open(file + base, O_RDONLY)) == -1) {
224     else if ((temp = open(file, O_RDONLY)) == -1) {
305         if (!sflag)
306             errmsg("grep: can't open %s\n", file);
307         nsucc = 2;
308         return;
309     }

311     /* read in first block of bytes */
312     if ((count = read(temp, prntbuf, GBUFSIZ)) <= 0) {
313         (void) close(temp);

315         if (cflag && !qflag) {
316             if (nfile > 1 && !hflag && file)
317                 (void) fprintf(stdout, "%s:", file);

```

```

318     if (!rflag)
319         (void) fprintf(stdout, "%lld\n", tln);
320     }
321     return;
322 }

324     lnum = 0;
325     ptr = prntbuf;
326     for (;;) {
327         /* look for next newline */
328         if ((ptrend = memchr(ptr + offset, '\n', count)) == NULL) {
329             offset += count;

331         /*
332          * shift unused data to the beginning of the buffer
333          */
334         if (ptr > prntbuf) {
335             (void) memmove(prntbuf, ptr, offset);
336             ptr = prntbuf;
337         }

339         /*
340          * re-allocate a larger buffer if this one is full
341          */
342         if (offset + GBUFSIZ > fw_lPrntBufLen) {
343             /*
344              * allocate a new buffer and preserve the
345              * contents...
346              */
347             fw_lPrntBufLen += GBUFSIZ;
348             if ((prntbuf = realloc(prntbuf,
349                 fw_lPrntBufLen)) == NULL)
350                 exit(2);

352             /*
353              * set up a bigger linebuffer (this is only used
354              * for case insensitive operations). Contents do
355              * not have to be preserved.
356              */
357             free(linebuf);
358             if ((linebuf = malloc(fw_lPrntBufLen)) == NULL)
359                 exit(2);

361             ptr = prntbuf;
362         }

364         p = prntbuf + offset;
365         if ((count = read(temp, p, GBUFSIZ)) > 0)
366             continue;

368         if (offset == 0)
369             /* end of file already reached */
370             break;

372         /* last line of file has no newline */
373         ptrend = ptr + offset;
374         nflag = 0;
375     } else {
376         next_ptr = ptrend + 1;
377         next_count = offset + count - (next_ptr - ptr);
378         nflag = 1;
379     }
380     lnum++;
381     *ptrend = '\0';

383     if (iflag) {

```

```

384         /*
385          * Make a lower case copy of the record
386          */
387         p = ptr;
388         for (lbuf = linebuf; p < ptrend; )
389             *lbuf++ = (char)tolower((int)
390                (unsigned char)*p++);
391         *lbuf = '\0';
392         lbuf = linebuf;
393     } else
394         /*
395          * Use record as is
396          */
397         lbuf = ptr;
398
399     /* lflag only once */
400     if ((step(lbuf, expbuf) ^ vflag) && succeed(file) == 1)
401         break;
402
403     if (!nlflag)
404         break;
405
406     ptr = next_ptr;
407     count = next_count;
408     offset = 0;
409 }
410 (void) close(temp);
411
412 if (cflag && !qflag) {
413     if (!hflag && file && (nfile > 1 ||
414        (rflag && outfn)))
415         if (nfile > 1 && !hflag && file)
416             (void) fprintf(stdout, "%s:", file);
417         (void) fprintf(stdout, "%lld\n", tln);
418 }
419
420 static int
421 succeed(const char *f)
422 {
423     int nchars;
424     nsucc = (nsucc == 2) ? 2 : 1;
425
426     if (f == NULL)
427         f = "<stdin>";
428
429     if (qflag) {
430         /* no need to continue */
431         return (1);
432     }
433
434     if (cflag) {
435         tln++;
436         return (0);
437     }
438
439     if (lflag) {
440         (void) fprintf(stdout, "%s\n", f);
441         return (1);
442     }
443
444     if (!hflag && (nfile > 1 || (rflag && outfn))) {
445         if (nfile > 1 && !hflag)
446             /* print filename */
447             (void) fprintf(stdout, "%s:", f);

```

```

447     }
448
449     if (bflag)
450         /* print block number */
451         (void) fprintf(stdout, "%lld:", (offset_t)
452            ((lseek(temp, (off_t)0, SEEK_CUR) - 1) / BLKSIZE));
453
454     if (nflag)
455         /* print line number */
456         (void) fprintf(stdout, "%lld:", lnum);
457
458     if (nlflag) {
459         /* newline at end of line */
460         *ptrend = '\n';
461         nchars = ptrend - ptr + 1;
462     } else {
463         /* don't write sentinel \0 */
464         nchars = ptrend - ptr;
465     }
466
467     (void) fwrite(ptr, 1, nchars, stdout);
468     return (0);
469 }

```

unchanged portion omitted

```

*****
27925 Mon Sep 17 14:32:33 2012
new/usr/src/cmd/grep_xpg4/grep.c
3047 grep support for -r would be useful
*****
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 2004 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  */

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

27 /*
28  * grep - pattern matching program - combined grep, egrep, and fgrep.
29  *   Based on MKS grep command, with XCU & Solaris mods.
30  */

32 /*
33  * Copyright 1985, 1992 by Mortice Kern Systems Inc. All rights reserved.
34  *
35  */

37 /* Copyright 2012 Nexenta Systems, Inc. All rights reserved. */

39 #include <string.h>
40 #include <stdlib.h>
41 #include <ctype.h>
42 #include <stdarg.h>
43 #include <regex.h>
44 #include <limits.h>
45 #include <sys/types.h>
46 #include <sys/stat.h>
47 #include <fcntl.h>
48 #include <stdio.h>
49 #include <locale.h>
50 #include <wchar.h>
51 #include <errno.h>
52 #include <unistd.h>
53 #include <wctype.h>
54 #include <ftw.h>
55 #include <sys/param.h>

57 #define BSIZE          512          /* Size of block for -b */
58 #define BUFSIZE        8192        /* Input buffer size */
59 #define MAX_DEPTH      1000        /* how deep to recurse */

```

```

61 #define M_CSETSIZE     256          /* singlebyte chars */
62 static int             bmglen;     /* length of BMG pattern */
63 static char            *bmgpat;    /* BMG pattern */
64 static int             bmgtab[M_CSETSIZE]; /* BMG delta table */

66 typedef struct         _PATTERN     {
67     char               *pattern;    /* original pattern */
68     wchar_t            *wmpattern;  /* wide, lowercased pattern */
69     struct _PATTERN    *next;      /* compiled pattern */
70     regex_t            re;
71 } PATTERN;

73 static PATTERN        *patterns;
74 static char            errstr[128]; /* regerror string buffer */
75 static int             regflags = 0; /* regcomp options */
76 static int             matched = 0; /* return of the grep() */
77 static int             errors = 0; /* count of errors */
78 static uchar_t        fgrep = 0;   /* Invoked as fgrep */
79 static uchar_t        egrep = 0;   /* Invoked as egrep */
80 static uchar_t        nvflag = 1;  /* Print matching lines */
81 static uchar_t        cflag;      /* Count of matches */
82 static uchar_t        iflag;      /* Case insensitive matching */
83 static uchar_t        hflag;      /* Suppress printing of filename */
84 static uchar_t        lflag;      /* Print file names of matches */
85 static uchar_t        nflag;      /* Precede lines by line number */
86 static uchar_t        rflag;      /* Search directories recursively */
87 static uchar_t        bflag;      /* Precede matches by block number */
88 static uchar_t        sflag;      /* Suppress file error messages */
89 static uchar_t        qflag;      /* Suppress standard output */
90 static uchar_t        wflag;      /* Search for expression as a word */
91 static uchar_t        xflag;      /* Anchoring */
92 static uchar_t        Eflag;      /* Egrep or -E flag */
93 static uchar_t        Fflag;      /* Fgrep or -F flag */
94 static uchar_t        Rflag;      /* Like rflag, but follow symlinks */
95 static uchar_t        outfn;      /* Put out file name */
96 static char            *cmdname;

98 static int             use_wchar, use_bmg, mblocale;

100 static size_t         outbuflen, prntbuflen;
101 static char            *prntbuf;
102 static wchar_t        *outline;

104 static void            addfile(const char *fn);
105 static void            addfile(char *fn);
106 static void            addpattern(char *s);
107 static void            fixpatterns(void);
108 static void            usage(void);
109 static int             grep(int, const char *);
110 static int             grep(int, char *);
111 static void            bmgcomp(char *, int);
112 static char            *bmgexec(char *, char *);
113 static int             recursive(const char *, const struct stat *, int, struct FTW *);
114 static void            process_path(const char *);
115 static void            process_file(const char *, int);

116 /*
117  * mainline for grep
118  */
119 int
120 main(int argc, char **argv)
121 {
122     char                *ap;
123     int                 matched = 0;
124     int                 c;

```

```

123     int     fflag = 0;
115     int     errors = 0;
124     int     i, n_pattern = 0, n_file = 0;
125     char    **pattern_list = NULL;
126     char    **file_list = NULL;

128     (void) setlocale(LC_ALL, "");
129 #if !defined(TEXT_DOMAIN) /* Should be defined by cc -D */
130 #define TEXT_DOMAIN      "SYS_TEST" /* Use this only if it weren't */
131 #endif
132     (void) textdomain(TEXT_DOMAIN);

134     /*
135      * true if this is running on the multibyte locale
136      */
137     mblocale = (MB_CUR_MAX > 1);
138     /*
139      * Skip leading slashes
140      */
141     cmdname = argv[0];
142     if (ap = strrchr(cmdname, '/'))
143         cmdname = ap + 1;

145     ap = cmdname;
146     /*
147      * Detect egrep/fgrep via command name, map to -E and -F options.
148      */
149     if (*ap == 'e' || *ap == 'E') {
150         regflags |= REG_EXTENDED;
151         egrep++;
152     } else {
153         if (*ap == 'f' || *ap == 'F') {
154             fgrep++;
155         }
156     }

158     while ((c = getopt(argc, argv, "vwchilnrbs:f:qxEFIR")) != EOF) {
159         while ((c = getopt(argc, argv, "vwchilnrbs:f:qxEFI")) != EOF) {
160             switch (c) {
161                 case 'v': /* POSIX: negate matches */
162                     nvflag = 0;
163                     break;

164                 case 'c': /* POSIX: write count */
165                     cflag++;
166                     break;

168                 case 'i': /* POSIX: ignore case */
169                     iflag++;
170                     regflags |= REG_ICASE;
171                     break;

173                 case 'l': /* POSIX: Write filenames only */
174                     lflag++;
175                     break;

177                 case 'n': /* POSIX: Write line numbers */
178                     nflag++;
179                     break;

181                 case 'r': /* Solaris: search recursively */
182                     rflag++;
183                     break;

185                 case 'b': /* Solaris: Write file block numbers */
186                     bflag++;

```

```

187         break;

189     case 's': /* POSIX: No error msgs for files */
190         sflag++;
191         break;

193     case 'e': /* POSIX: pattern list */
194         n_pattern++;
195         pattern_list = realloc(pattern_list,
196             sizeof(char *) * n_pattern);
197         if (pattern_list == NULL) {
198             (void) fprintf(stderr,
199                 gettext("%s: out of memory\n"),
200                 cmdname);
201             exit(2);
202         }
203         *(pattern_list + n_pattern - 1) = optarg;
204         break;

206     case 'f': /* POSIX: pattern file */
207         fflag = 1;
208         n_file++;
209         file_list = realloc(file_list,
210             sizeof(char *) * n_file);
211         if (file_list == NULL) {
212             (void) fprintf(stderr,
213                 gettext("%s: out of memory\n"),
214                 cmdname);
215             exit(2);
216         }
217         *(file_list + n_file - 1) = optarg;
218         break;
219     case 'h': /* Solaris: suppress printing of file name */
220         hflag = 1;
221         break;

223     case 'q': /* POSIX: quiet: status only */
224         qflag++;
225         break;

227     case 'w': /* Solaris: treat pattern as word */
228         wflag++;
229         break;

231     case 'x': /* POSIX: full line matches */
232         xflag++;
233         regflags |= REG_ANCHOR;
234         break;

236     case 'E': /* POSIX: Extended RE's */
237         regflags |= REG_EXTENDED;
238         Eflag++;
239         break;

241     case 'F': /* POSIX: strings, not RE's */
242         Fflag++;
243         break;

245     case 'R': /* Solaris: like rflag, but follow symlinks */
246         Rflag++;
247         rflag++;
248         break;

250     default:
251         usage();
252     }

```

```

253     }
254     /*
255     * If we're invoked as egrep or fgrep we need to do some checks
256     */

258     if (egrep || fgrep) {
259         /*
260         * Use of -E or -F with egrep or fgrep is illegal
261         */
262         if (Eflag || Fflag)
263             usage();
264         /*
265         * Don't allow use of wflag with egrep / fgrep
266         */
267         if (wflag)
268             usage();
269         /*
270         * For Solaris the -s flag is equivalent to XCU -q
271         */
272         if (sflag)
273             qflag++;
274         /*
275         * done with above checks - set the appropriate flags
276         */
277         if (egrep)
278             Eflag++;
279         else
280             Fflag++;          /* Else fgrep */
281     }

283     if (wflag && (Eflag || Fflag)) {
284         /*
285         * -w cannot be specified with grep -F
286         */
287         usage();
288     }

290     /*
291     * -E and -F flags are mutually exclusive - check for this
292     */
293     if (Eflag && Fflag)
294         usage();

296     /*
297     * -c, -l and -q flags are mutually exclusive
298     * We have -c override -l like in Solaris.
299     * -q overrides -l & -c programmatically in grep() function.
300     */
301     if (cflag && lflag)
302         lflag = 0;

304     argv += optind - 1;
305     argc -= optind - 1;

307     /*
308     * Now handling -e and -f option
309     */
310     if (pattern_list) {
311         for (i = 0; i < n_pattern; i++) {
312             addpattern(pattern_list[i]);
313         }
314         free(pattern_list);
315     }
316     if (file_list) {
317         for (i = 0; i < n_file; i++) {
318             addfile(file_list[i]);

```

```

319     }
320     free(file_list);
321 }

323     /*
324     * No -e or -f? Make sure there is one more arg, use it as the pattern.
325     */
326     if (patterns == NULL && !fflag) {
327         if (argc < 2)
328             usage();
329         addpattern(argv[1]);
330         argc--;
331         argv++;
332     }

334     /*
335     * If -x flag is not specified or -i flag is specified
336     * with fgrep in a multibyte locale, need to use
337     * the wide character APIs. Otherwise, byte-oriented
338     * process will be done.
339     */
340     use_wchar = Fflag && mblocale && (!xflag || iflag);

342     /*
343     * Compile Patterns and also decide if BMG can be used
344     */
345     fixpatterns();

347     /* Process all files: stdin, or rest of arg list */
348     if (argc < 2) {
349         matched = grep(0, gettext("(standard input)"));
350     } else {
351         if (argc > 2 && hflag == 0)
352             outfn = 1;          /* Print filename on match line */
353         for (argv++; *argv != NULL; argv++) {
354             process_path(*argv);
355             int fd;

356             if ((fd = open(*argv, O_RDONLY)) == -1) {
357                 errors = 1;
358                 if (sflag)
359                     continue;
360                 (void) fprintf(stderr, gettext(
361                     "%s: can't open \"%s\"\n"),
362                     cmdname, *argv);
363                 continue;
364             }
365             matched |= grep(fd, *argv);
366             (void) close(fd);
367             if (ferror(stdout))
368                 break;
369         }
370     }

372     /*
373     * Return() here is used instead of exit
374     */

376     (void) fflush(stdout);

378     if (errors)
379         return (2);
380     return (matched ? 0 : 1);
381 }

383 static void
384 process_path(const char *path)

```

```

370 {
371     struct stat st;
372     int walkflags = FTW_CHDIR;
373     char *buf = NULL;

375     if (rflag) {
376         if (stat(path, &st) != -1 &&
377             (st.st_mode & S_IFMT) == S_IFDIR) {
378             outfn = 1; /* Print filename */

380             /*
381              * Add trailing slash if arg
382              * is directory, to resolve symlinks.
383              */
384             if (path[strlen(path) - 1] != '/') {
385                 (void) asprintf(&buf, "%s/", path);
386                 if (buf != NULL)
387                     path = buf;
388             }

390             /*
391              * Search through subdirs if path is directory.
392              * Don't follow symlinks if Rflag is not set.
393              */
394             if (!Rflag)
395                 walkflags |= FTW_PHYS;

397             if (nftw(path, recursive, MAX_DEPTH, walkflags) != 0) {
398                 if (!sflag)
399                     (void) fprintf(stderr,
400                         gettext("%s: can't open \"%s\"\n"),
401                         cmdname, path);
402                 errors = 1;
403             }
404             return;
405         }
406     }
407     process_file(path, 0);
408 }

410 /*
411  * Read and process all files in directory recursively.
412  */
413 static int
414 recursive(const char *name, const struct stat *statp, int info, struct FTW *ftw)
415 {
416     /*
417      * Process files and follow symlinks if Rflag set.
418      */
419     if (info != FTW_F) {
420         /* Report broken symlinks and unreadable files */
421         if (!sflag &&
422             (info == FTW_SLN || info == FTW_DNR || info == FTW_NS)) {
423             (void) fprintf(stderr,
424                 gettext("%s: can't open \"%s\"\n"), cmdname, name);
425         }
426         return (0);
427     }

430     /* Skip devices and pipes if Rflag is not set */
431     if (!Rflag && !S_ISREG(statp->st_mode))
432         return (0);
433     /* Pass offset to relative name from FTW_CHDIR */
434     process_file(name, ftw->base);
435     return (0);

```

```

436 }

438 /*
439  * Opens file and call grep function.
440  */
441 static void
442 process_file(const char *name, int base)
443 {
444     int fd;

446     if ((fd = open(name + base, O_RDONLY)) == -1) {
447         errors = 1;
448         if (!sflag) /* Silent mode */
449             (void) fprintf(stderr, gettext(
450                 "%s: can't open \"%s\"\n"),
451                 cmdname, name);
452         return;
453     }
454     matched |= grep(fd, name);
455     (void) close(fd);

457     if (ferror(stdout)) {
458         (void) fprintf(stderr, gettext(
459             "%s: error writing to stdout\n"),
460             cmdname);
461         (void) fflush(stdout);
462         exit(2);
463     }

465 }

467 /*
468  * Add a file of strings to the pattern list.
469  */
470 static void
471 addfile(const char *fn)
472 {
473     FILE *fp;
474     char *inbuf;
475     char *bufp;
476     size_t bufsiz, buflen, bufused;

478     /*
479      * Open the pattern file
480      */
481     if ((fp = fopen(fn, "r")) == NULL) {
482         (void) fprintf(stderr, gettext("%s: can't open \"%s\"\n"),
483             cmdname, fn);
484         exit(2);
485     }
486     bufsiz = BUFSIZE;
487     if ((inbuf = malloc(bufsiz)) == NULL) {
488         (void) fprintf(stderr,
489             gettext("%s: out of memory\n"), cmdname);
490         exit(2);
491     }
492     bufp = inbuf;
493     bufused = 0;
494     /*
495      * Read in the file, reallocating as we need more memory
496      */
497     while (fgets(bufp, bufsiz - bufused, fp) != NULL) {
498         buflen = strlen(bufp);
499         bufused += buflen;
500         if (bufused + 1 == bufsiz && bufp[buflen - 1] != '\n') {

```



```

501      /*
502      * if this line does not fit to the buffer,
503      * realloc larger buffer
504      */
505      bufsiz += BUFSIZE;
506      if ((inbuf = realloc(inbuf, bufsiz)) == NULL) {
507          (void) fprintf(stderr,
508              gettext("%s: out of memory\n"),
509              cmdname);
510          exit(2);
511      }
512      bufp = inbuf + bufused;
513      continue;
514  }
515  if (bufp[buflen - 1] == '\n') {
516      bufp[--buflen] = '\0';
517  }
518  addpattern(inbuf);

520      bufp = inbuf;
521      bufused = 0;
522  }
523  free(inbuf);
524  (void) fclose(fp);
525 }

```

unchanged portion omitted

```

767 /*
768 * Do grep on a single file.
769 * Return true in any lines matched.
770 *
771 * We have two strategies:
772 * The fast one is used when we have a single pattern with
773 * a string known to occur in the pattern. We can then
774 * do a BMG match on the whole buffer.
775 * This is an order of magnitude faster.
776 * Otherwise we split the buffer into lines,
777 * and check for a match on each line.
778 */
779 static int
780 grep(int fd, const char *fn)
781 {
782     PATTERN *pp;
783     off_t data_len; /* length of the data chunk */
784     off_t line_len; /* length of the current line */
785     off_t line_offset; /* current line's offset from the beginning */
786     long long lineno;
787     long long matches = 0; /* Number of matching lines */
788     int newlinep; /* 0 if the last line of file has no newline */
789     char *ptr, *ptrend;

792     if (patterns == NULL)
793         return (0); /* no patterns to match -- just return */

795     pp = patterns;

797     if (use_bmg) {
798         bmgcomp(pp->pattern, strlen(pp->pattern));
799     }

801     if (use_wchar && outline == NULL) {
802         outbuflen = BUFSIZE + 1;
803         outline = malloc(sizeof(wchar_t) * outbuflen);
804         if (outline == NULL) {

```

```

805         (void) fprintf(stderr, gettext("%s: out of memory\n"),
806             cmdname);
807         exit(2);
808     }
809 }

811     if (prntbuf == NULL) {
812         prntbuflen = BUFSIZE;
813         if ((prntbuf = malloc(prntbuflen + 1)) == NULL) {
814             (void) fprintf(stderr, gettext("%s: out of memory\n"),
815                 cmdname);
816             exit(2);
817         }
818     }

820     line_offset = 0;
821     lineno = 0;
822     newlinep = 1;
823     data_len = 0;
824     for (; ; ) {
825         long count;
826         off_t offset = 0;

828         if (data_len == 0) {
829             /*
830              * If no data in the buffer, reset ptr
831              */
832             ptr = prntbuf;
833         }
834         if (ptr == prntbuf) {
835             /*
836              * The current data chunk starts from prntbuf.
837              * This means either the buffer has no data
838              * or the buffer has no newline.
839              * So, read more data from input.
840              */
841             count = read(fd, ptr + data_len, prntbuflen - data_len);
842             if (count < 0) {
843                 /* read error */
844                 if (cflag) {
845                     if (outfn && !rflag) {
846                         if (outfn) {
847                             (void) fprintf(stdout,
848                                 "%s:", fn);
849                         }
850                     }
851                     if (!qflag && !rflag) {
852                         if (!qflag) {
853                             (void) fprintf(stdout, "%lld\n",
854                                 matches);
855                         }
856                     }
857                     return (0);
858                 } else if (count == 0) {
859                     /* no new data */
860                     if (data_len == 0) {
861                         /* end of file already reached */
862                         break;
863                     }
864                     /* last line of file has no newline */
865                     ptr = ptr + data_len;
866                     newlinep = 0;
867                     goto L_start_process;
868                 }
869             }
870             offset = data_len;
871             data_len += count;
872         }
873     }

```

```

870     /*
871     * Look for newline in the chunk
872     * between ptr + offset and ptr + data_len - offset.
873     */
874     ptrend = find_nl(ptr + offset, data_len - offset);
875     if (ptrend == NULL) {
876         /* no newline found in this chunk */
877         if (ptr > prntbuf) {
878             /*
879              * Move remaining data to the beginning
880              * of the buffer.
881              * Remaining data lie from ptr for
882              * data_len bytes.
883              */
884             (void) memmove(prntbuf, ptr, data_len);
885         }
886         if (data_len == prntbuflen) {
887             /*
888              * No enough room in the buffer
889              */
890             prntbuflen += BUFSIZE;
891             prntbuf = realloc(prntbuf, prntbuflen + 1);
892             if (prntbuf == NULL) {
893                 (void) fprintf(stderr,
894                     gettext("%s: out of memory\n"),
895                     cmdname);
896                 exit(2);
897             }
898         }
899         ptr = prntbuf;
900         /* read the next input */
901         continue;
902     }
903 L_start_process:
904
905     /*
906     * Beginning of the chunk:      ptr
907     * End of the chunk:          ptr + data_len
908     * Beginning of the line:      ptr
909     * End of the line:           ptrend
910     */
911
912     if (use_bmg) {
913         /*
914         * Use Boyer-Moore-Gosper algorithm to find out if
915         * this chunk (not this line) contains the specified
916         * pattern. If not, restart from the last line
917         * of this chunk.
918         */
919         char *bline;
920         bline = bmgexec(ptr, ptr + data_len);
921         if (bline == NULL) {
922             /*
923              * No pattern found in this chunk.
924              * Need to find the last line
925              * in this chunk.
926              */
927             ptrend = rfind_nl(ptr, data_len);
928
929             /*
930              * When this chunk does not contain newline,
931              * ptrend becomes NULL, which should happen
932              * when the last line of file does not end
933              * with a newline. At such a point,
934              * newlinep should have been set to 0.

```

```

935         * Therefore, just after jumping to
936         * L_skip_line, the main for-loop quits,
937         * and the line_len value won't be
938         * used.
939         */
940         line_len = ptrend - ptr;
941         goto L_skip_line;
942     }
943     if (bline > ptrend) {
944         /*
945          * Pattern found not in the first line
946          * of this chunk.
947          * Discard the first line.
948          */
949         line_len = ptrend - ptr;
950         goto L_skip_line;
951     }
952     /*
953     * Pattern found in the first line of this chunk.
954     * Using this result.
955     */
956     *ptrend = '\0';
957     line_len = ptrend - ptr;
958
959     /*
960     * before jumping to L_next_line,
961     * need to handle xflag if specified
962     */
963     if (xflag && (line_len != bmglen ||
964         strcmp(bmgpat, ptr) != 0)) {
965         /* didn't match */
966         pp = NULL;
967     } else {
968         pp = patterns; /* to make it happen */
969     }
970     goto L_next_line;
971 }
972 lineneno++;
973 /*
974 * Line starts from ptr and ends at ptrend.
975 * line_len will be the length of the line.
976 */
977 *ptrend = '\0';
978 line_len = ptrend - ptr;
979
980 /*
981 * From now, the process will be performed based
982 * on the line from ptr to ptrend.
983 */
984 if (use_wchar) {
985     size_t len;
986
987     if (line_len >= outbuflen) {
988         outbuflen = line_len + 1;
989         outline = realloc(outline,
990             sizeof(wchar_t) * outbuflen);
991         if (outline == NULL) {
992             (void) fprintf(stderr,
993                 gettext("%s: out of memory\n"),
994                 cmdname);
995             exit(2);
996         }
997     }
998
999     len = mbstowcs(outline, ptr, line_len);
1000     if (len == (size_t)-1) {

```

```

1001         (void) fprintf(stderr, gettext(
1002     "%s: input file \"%s\": line %lld: invalid multibyte character\n"),
1003         cmdname, fn, lineno);
1004         /* never match a line with invalid sequence */
1005         goto L_skip_line;
1006     }
1007     outline[len] = L'\0';
1008
1009     if (iflag) {
1010         wchar_t *cp;
1011         for (cp = outline; *cp != '\0'; cp++) {
1012             *cp = towlower((wint_t)*cp);
1013         }
1014     }
1015
1016     if (xflag) {
1017         for (pp = patterns; pp; pp = pp->next) {
1018             if (outline[0] == pp->wpattern[0] &&
1019                 wcsncmp(outline,
1020                     pp->wpattern) == 0) {
1021                 /* matched */
1022                 break;
1023             }
1024         }
1025     } else {
1026         for (pp = patterns; pp; pp = pp->next) {
1027             if (wcsncmp(outline, pp->wpattern)
1028                 != NULL) {
1029                 /* matched */
1030                 break;
1031             }
1032         }
1033     }
1034 } else if (Fflag) {
1035     /* fgrep in byte-oriented handling */
1036     char *fptr;
1037     if (iflag) {
1038         fptr = istrdup(ptr);
1039     } else {
1040         fptr = ptr;
1041     }
1042     if (xflag) {
1043         /* fgrep -x */
1044         for (pp = patterns; pp; pp = pp->next) {
1045             if (fptr[0] == pp->pattern[0] &&
1046                 strcmp(fptr, pp->pattern) == 0) {
1047                 /* matched */
1048                 break;
1049             }
1050         }
1051     } else {
1052         for (pp = patterns; pp; pp = pp->next) {
1053             if (strstr(fptr, pp->pattern) != NULL) {
1054                 /* matched */
1055                 break;
1056             }
1057         }
1058     }
1059 } else {
1060     /* grep or egrep */
1061     for (pp = patterns; pp; pp = pp->next) {
1062         int rv;
1063
1064         rv = regexec(&pp->re, ptr, 0, NULL, 0);
1065         if (rv == REG_OK) {
1066             /* matched */

```

```

1067         break;
1068     }
1069
1070     switch (rv) {
1071     case REG_NOMATCH:
1072         break;
1073     case REG_ECHAR:
1074         (void) fprintf(stderr, gettext(
1075     "%s: input file \"%s\": line %lld: invalid multibyte character\n"),
1076         cmdname, fn, lineno);
1077         break;
1078     default:
1079         (void) regerror(rv, &pp->re, errstr,
1080             sizeof (errstr));
1081         (void) fprintf(stderr, gettext(
1082     "%s: input file \"%s\": line %lld: %s\n"),
1083             cmdname, fn, lineno, errstr);
1084         exit(2);
1085     }
1086 }
1087
1088 }
1089
1090 L_next_line:
1091     /*
1092     * Here, if pp points to non-NULL, something has been matched
1093     * to the pattern.
1094     */
1095     if (nvflag == (pp != NULL)) {
1096         matches++;
1097         /*
1098         * Handle q, l, and c flags.
1099         */
1100         if (qflag) {
1101             /* no need to continue */
1102             /*
1103             * End of this line is ptrend.
1104             * We have read up to ptr + data_len.
1105             */
1106             off_t pos;
1107             pos = ptr + data_len - (ptrend + 1);
1108             (void) lseek(fd, -pos, SEEK_CUR);
1109             exit(0);
1110         }
1111         if (lflag) {
1112             (void) printf("%s\n", fn);
1113             break;
1114         }
1115         if (!cflag) {
1116             if (outfn) {
1117                 (void) printf("%s:", fn);
1118             }
1119             if (bflag) {
1120                 (void) printf("%lld:", (offset_t)
1121                     (line_offset / BSIZE));
1122             }
1123             if (nflag) {
1124                 (void) printf("%lld:", lineno);
1125             }
1126             *ptrend = '\n';
1127             (void) fwrite(ptr, 1, line_len + 1, stdout);
1128         }
1129         if (ferror(stdout)) {
1130             return (0);
1131         }
1132     }
1133
1134 L_skip_line:

```

```

1133         if (!newlinep)
1134             break;

1136     data_len -= line_len + 1;
1137     line_offset += line_len + 1;
1138     ptr = ptrend + 1;
1139 }

1141 if (cflag) {
1142     if (outfn) {
1143         (void) printf("%s:", fn);
1144     }
1145     if (!qflag) {
1146         (void) printf("%lld\n", matches);
1147     }
1148 }
1149 return (matches != 0);
1150 }

1152 /*
1153  * usage message for grep
1154  */
1155 static void
1156 usage(void)
1157 {
1158     if (egrep || fgrep) {
1159         (void) fprintf(stderr, gettext("Usage:\t%s"), cmdname);
1160         (void) fprintf(stderr,
1161             gettext(" [-c|-l|-q] [-r|-R] [-bhinsvx] "
1162                 gettext(" [-c|-l|-q] [-bhinsvx] "
1163                     "pattern_list [file ...]\n"));
1164     }
1165     (void) fprintf(stderr, "\t%s", cmdname);
1166     (void) fprintf(stderr,
1167         gettext(" [-c|-l|-q] [-r|-R] [-bhinsvx] "
1168             "[ -e pattern_list]... "
1169             "[-f pattern_file]... [file...]\n"));
1170 } else {
1171     (void) fprintf(stderr, gettext("Usage:\t%s"), cmdname);
1172     (void) fprintf(stderr,
1173         gettext(" [-c|-l|-q] [-r|-R] [-bhinsvwx] "
1174             gettext(" [-c|-l|-q] [-bhinsvwx] "
1175                 "pattern_list [file ...]\n"));
1176     (void) fprintf(stderr, "\t%s", cmdname);
1177     (void) fprintf(stderr,
1178         gettext(" [-c|-l|-q] [-r|-R] [-bhinsvwx] "
1179             "[ -e pattern_list]... "
1180             "[-f pattern_file]... [file...]\n"));
1181     (void) fprintf(stderr, "\t%s", cmdname);
1182     (void) fprintf(stderr,
1183         gettext(" -E [-c|-l|-q] [-r|-R] [-bhinsvx] "
1184             gettext(" -E [-c|-l|-q] [-bhinsvx] "
1185                 "pattern_list [file ...]\n"));
1186     (void) fprintf(stderr, "\t%s", cmdname);
1187     (void) fprintf(stderr,
1188         gettext(" -E [-c|-l|-q] [-r|-R] [-bhinsvx] "
1189             "[ -e pattern_list]... "
1190             "[-f pattern_file]... [file...]\n"));
1191     (void) fprintf(stderr, "\t%s", cmdname);

```

```

1193         (void) fprintf(stderr,
1194             gettext(" -F [-c|-l|-q] [-r|-R] [-bhinsvx] "
1195                 gettext(" -F [-c|-l|-q] [-bhinsvx] "
1196                     "pattern_list [file ...]\n"));
1197     (void) fprintf(stderr, "\t%s", cmdname);
1198     (void) fprintf(stderr,
1199         gettext(" -F [-c|-l|-q] [-bhinsvx] [-e pattern_list]... "
1200             "[-f pattern_file]... [file...]\n"));
1201     }
1202     exit(2);
1203     /* NOTREACHED */
1204 }

```

unchanged_portion_omitted

```

*****
13910 Mon Sep 17 14:32:33 2012
new/usr/src/man/man1/grep.1
3047 grep support for -r would be useful
*****
1 \" te
2 .\" Copyright 2012 Nexenta Systems, Inc. All rights reserved.
3 .\" Copyright 1989 AT&T
4 .\" Copyright (c) 2008, Sun Microsystems, Inc. All Rights Reserved
5 .\" Portions Copyright (c) 1992, X/Open Company Limited All Rights Reserved
6 .\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
7 .\" http://www.opengroup.org/bookstore/.
8 .\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
9 .\" This notice shall appear on any product containing this material.
10 .\" The contents of this file are subject to the terms of the Common Development
11 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
12 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
13 .TH GREP 1 "Feb 26, 2008"
14 .SH NAME
15 grep \- search a file for a pattern
16 .SH SYNOPSIS
17 .LP
18 .nf
19 \fB/usr/bin/grep\fR [\fB-c\fR | \fB-l\fR | \fB-q\fR] [\fB-r\fR | \fB-R\fR] [\fB-b
20 \fIlimited-regular-expression\fR [\fIfilename\fR]...
19 \fB/usr/bin/grep\fR [\fB-c\fR | \fB-l\fR | \fB-q\fR] [\fB-bhinsvw\fR] \fIlimited
20 [\fIfilename\fR]...
21 .fi
23 .LP
24 .nf
25 \fB/usr/xpg4/bin/grep\fR [\fB-E\fR | \fB-F\fR] [\fB-c\fR | \fB-l\fR | \fB-q\fR]
26 [\fB-bhinsvw\fR] \fB-e\fR \fIpattern_list\fR... [\fB-f\fR \fIpattern_file\fR]
27 [\fIfile\fR]...
25 \fB/usr/xpg4/bin/grep\fR [\fB-E\fR | \fB-F\fR] [\fB-c\fR | \fB-l\fR | \fB-q\fR]
26 [\fB-f\fR \fIpattern_file\fR]... [\fIfile\fR]...
28 .fi
30 .LP
31 .nf
32 \fB/usr/xpg4/bin/grep\fR [\fB-E\fR | \fB-F\fR] [\fB-c\fR | \fB-l\fR | \fB-q\fR]
33 [\fB-bhinsvw\fR] \fB-e\fR \fIpattern_list\fR... \fB-f\fR \fIpattern_file\
34 [\fIfile\fR]...
31 \fB/usr/xpg4/bin/grep\fR [\fB-E\fR | \fB-F\fR] [\fB-c\fR | \fB-l\fR | \fB-q\fR]
32 [\fB-e\fR \fIpattern_list\fR]... \fB-f\fR \fIpattern_file\fR... [\fIfile\fR
35 .fi
37 .LP
38 .nf
39 \fB/usr/xpg4/bin/grep\fR [\fB-E\fR | \fB-F\fR] [\fB-c\fR | \fB-l\fR | \fB-q\fR]
40 [\fB-bhinsvw\fR] \fIpattern\fR [\fIfile\fR]...
37 \fB/usr/xpg4/bin/grep\fR [\fB-E\fR | \fB-F\fR] [\fB-c\fR | \fB-l\fR | \fB-q\fR]
38 [\fIfile\fR]...
41 .fi
43 .SH DESCRIPTION
44 .sp
45 .LP
46 The \fBgrep\fR utility searches text files for a pattern and prints all lines
47 that contain that pattern. It uses a compact non-deterministic algorithm.
48 .sp
49 .LP
50 Be careful using the characters \fB$\fR, \fB*\fR, \fB[\fR, \fB^\fR, \fB|\fR,
51 \fB(\fR, \fB)\fR, and \fB\e\fR in the \fIpattern_list\fR because they are also
52 meaningful to the shell. It is safest to enclose the entire \fIpattern_list\fR
53 in single quotes \fBa'\fR&...\fBa'\fR&.

```

```

54 .sp
55 .LP
56 If no files are specified, \fBgrep\fR assumes standard input. Normally, each
57 line found is copied to standard output. The file name is printed before each
58 line found if there is more than one input file.
59 .SS "/usr/bin/grep"
60 .sp
61 .LP
62 The \fB/usr/bin/grep\fR utility uses limited regular expressions like those
63 described on the \fBregexp\fR(5) manual page to match the patterns.
64 .SS "/usr/xpg4/bin/grep"
65 .sp
66 .LP
67 The options \fB-E\fR and \fB-F\fR affect the way \fB/usr/xpg4/bin/grep\fR
68 interprets \fIpattern_list\fR. If \fB-E\fR is specified,
69 \fB/usr/xpg4/bin/grep\fR interprets \fIpattern_list\fR as a full regular
70 expression (see \fB-E\fR for description). If \fB-F\fR is specified,
71 \fBgrep\fR interprets \fIpattern_list\fR as a fixed string. If neither are
72 specified, \fBgrep\fR interprets \fIpattern_list\fR as a basic regular
73 expression as described on \fBregex\fR(5) manual page.
74 .SH OPTIONS
75 .sp
76 .LP
77 The following options are supported for both \fB/usr/bin/grep\fR and
78 \fB/usr/xpg4/bin/grep\fR:
79 .sp
80 .ne 2
81 .na
82 \fB\b\fR
83 .ad
84 .RS 6n
85 Precedes each line by the block number on which it was found. This can be
86 useful in locating block numbers by context (first block is 0).
87 .RE
89 .sp
90 .ne 2
91 .na
92 \fB\c\fR
93 .ad
94 .RS 6n
95 Prints only a count of the lines that contain the pattern.
96 .RE
98 .sp
99 .ne 2
100 .na
101 \fB\h\fR
102 .ad
103 .RS 6n
104 Prevents the name of the file containing the matching line from being prepended
105 to that line. Used when searching multiple files.
106 .RE
108 .sp
109 .ne 2
110 .na
111 \fB\i\fR
112 .ad
113 .RS 6n
114 Ignores upper/lower case distinction during comparisons.
115 .RE
117 .sp
118 .ne 2
119 .na

```

```

120 \fB\fB-l\fR\fR
121 .ad
122 .RS 6n
123 Prints only the names of files with matching lines, separated by NEWLINE
124 characters. Does not repeat the names of files when the pattern is found more
125 than once.
126 .RE

```

```

128 .sp
129 .ne 2
130 .na
131 \fB\fB-n\fR\fR
132 .ad
133 .RS 6n
134 Precedes each line by its line number in the file (first line is 1).
135 .RE

```

```

137 .sp
138 .ne 2
139 .na
140 \fB\fB-r\fR\fR
141 .ad
142 .RS 6n
143 Read all files under each directory, recursively. Follow symbolic links on
144 the command line, but skip symlinks that are encountered recursively. If file
145 is a device, FIFO, or socket, skip it.
146 .RE

```

```

148 .sp
149 .ne 2
150 .na
151 \fB\fB-R\fR\fR
152 .ad
153 .RS 6n
154 Read all files under each directory, recursively, following all symbolic links.
155 .RE

```

```

157 .sp
158 .ne 2
159 .na
160 \fB\fB-q\fR\fR
161 .ad
162 .RS 6n
163 Quiet. Does not write anything to the standard output, regardless of matching
164 lines. Exits with zero status if an input line is selected.
165 .RE

```

```

167 .sp
168 .ne 2
169 .na
170 \fB\fB-s\fR\fR
171 .ad
172 .RS 6n
173 Suppresses error messages about nonexistent or unreadable files.
174 .RE

```

```

176 .sp
177 .ne 2
178 .na
179 \fB\fB-v\fR\fR
180 .ad
181 .RS 6n
182 Prints all lines except those that contain the pattern.
183 .RE

```

```

185 .sp

```

```

186 .ne 2
187 .na
188 \fB\fB-w\fR\fR
189 .ad
190 .RS 6n
191 Searches for the expression as a word as if surrounded by \fB\e<\fR and
192 \fB\e>\fR&.
193 .RE

```

```

195 .SS "/usr/xpg4/bin/grep"
196 .sp
197 .LP
198 The following options are supported for \fB/usr/xpg4/bin/grep\fR only:
199 .sp
200 .ne 2
201 .na
202 \fB\fB-e\fR \fIpattern_list\fR\fR
203 .ad
204 .RS 19n
205 Specifies one or more patterns to be used during the search for input. Patterns
206 in \fIpattern_list\fR must be separated by a NEWLINE character. A null pattern
207 can be specified by two adjacent newline characters in \fIpattern_list\fR.
208 Unless the \fB-E\fR or \fB-F\fR option is also specified, each pattern is
209 treated as a basic regular expression. Multiple \fB-e\fR and \fB-f\fR options
210 are accepted by \fBgrep\fR. All of the specified patterns are used when
211 matching lines, but the order of evaluation is unspecified.
212 .RE

```

```

214 .sp
215 .ne 2
216 .na
217 \fB\fB-E\fR\fR
218 .ad
219 .RS 19n
220 Matches using full regular expressions. Treats each pattern specified as a full
221 regular expression. If any entire full regular expression pattern matches an
222 input line, the line is matched. A null full regular expression matches every
223 line. Each pattern is interpreted as a full regular expression as described on
224 the \fBregex\fR(5) manual page, except for \fB\e(\fR and \fB\e)\fR, and
225 including:
226 .RS +4
227 .TP
228 1.
229 A full regular expression followed by \fB+\fR that matches one or more
230 occurrences of the full regular expression.
231 .RE
232 .RS +4
233 .TP
234 2.
235 A full regular expression followed by \fB?\fR that matches 0 or 1
236 occurrences of the full regular expression.
237 .RE
238 .RS +4
239 .TP
240 3.
241 Full regular expressions separated by | or by a new-line that match strings
242 that are matched by any of the expressions.
243 .RE
244 .RS +4
245 .TP
246 4.
247 A full regular expression that is enclosed in parentheses \fB()\fR for
248 grouping.
249 .RE
250 The order of precedence of operators is \fB[|\]\fR, then \fB*\fR, \fB|\fR, then
251 concatenation, then | and new-line.

```

```

252 .RE
254 .sp
255 .ne 2
256 .na
257 \fB\fB-f\fR \fIpattern_file\fR\fR
258 .ad
259 .RS 19n
260 Reads one or more patterns from the file named by the path name
261 \fIpattern_file\fR. Patterns in \fIpattern_file\fR are terminated by a NEWLINE
262 character. A null pattern can be specified by an empty line in
263 \fIpattern_file\fR. Unless the \fB-E\fR or \fB-F\fR option is also specified,
264 each pattern is treated as a basic regular expression.
265 .RE
267 .sp
268 .ne 2
269 .na
270 \fB\F\fR
271 .ad
272 .RS 19n
273 Matches using fixed strings. Treats each pattern specified as a string instead
274 of a regular expression. If an input line contains any of the patterns as a
275 contiguous sequence of bytes, the line is matched. A null string matches every
276 line. See \fBfgrep\fR(1) for more information.
277 .RE
279 .sp
280 .ne 2
281 .na
282 \fB\fB-x\fR\fR
283 .ad
284 .RS 19n
285 Considers only input lines that use all characters in the line to match an
286 entire fixed string or regular expression to be matching lines.
287 .RE
289 .SH OPERANDS
290 .sp
291 .LP
292 The following operands are supported:
293 .sp
294 .ne 2
295 .na
296 \fB\fIfile\fR\fR
297 .ad
298 .RS 8n
299 A path name of a file to be searched for the patterns. If no \fIfile\fR
300 operands are specified, the standard input is used.
301 .RE
303 .SS "/usr/bin/grep"
304 .sp
305 .ne 2
306 .na
307 \fB\fIpattern\fR\fR
308 .ad
309 .RS 11n
310 Specifies a pattern to be used during the search for input.
311 .RE
313 .SS "/usr/xpg4/bin/grep"
314 .sp
315 .ne 2
316 .na
317 \fB\fIpattern\fR\fR

```

```

318 .ad
319 .RS 11n
320 Specifies one or more patterns to be used during the search for input. This
321 operand is treated as if it were specified as \fB-e\fR \fIpattern_list\fR.
322 .RE
324 .SH USAGE
325 .sp
326 .LP
327 The \fB-e\fR \fIpattern_list\fR option has the same effect as the
328 \fIpattern_list\fR operand, but is useful when \fIpattern_list\fR begins with
329 the hyphen delimiter. It is also useful when it is more convenient to provide
330 multiple patterns as separate arguments.
331 .sp
332 .LP
333 Multiple \fB-e\fR and \fB-f\fR options are accepted and \fBgrep\fR uses all of
334 the patterns it is given while matching input text lines. Notice that the order
335 of evaluation is not specified. If an implementation finds a null string as a
336 pattern, it is allowed to use that pattern first, matching every line, and
337 effectively ignore any other patterns.
338 .sp
339 .LP
340 The \fB-q\fR option provides a means of easily determining whether or not a
341 pattern (or string) exists in a group of files. When searching several files,
342 it provides a performance improvement (because it can quit as soon as it finds
343 the first match) and requires less care by the user in choosing the set of
344 files to supply as arguments (because it exits zero if it finds a match even if
345 \fBgrep\fR detected an access or read error on earlier file operands).
346 .SS "Large File Behavior"
347 .sp
348 .LP
349 See \fBlargefile\fR(5) for the description of the behavior of \fBgrep\fR when
350 encountering files greater than or equal to 2 Gbyte ( 2^31 bytes).
351 .SH EXAMPLES
352 .LP
353 \fBExample 1 \fRFinding All Uses of a Word
354 .sp
355 .LP
356 To find all uses of the word "\fBposix\fR" (in any case) in the file
357 \fBtext.mm\fR, and write with line numbers:
359 .sp
360 .in +2
361 .nf
362 example% \fB/usr/bin/grep -i -n posix text.mm\fR
363 .fi
364 .in -2
365 .sp
367 .LP
368 \fBExample 2 \fRFinding All Empty Lines
369 .sp
370 .LP
371 To find all empty lines in the standard input:
373 .sp
374 .in +2
375 .nf
376 example% \fB/usr/bin/grep ^$\fR
377 .fi
378 .in -2
379 .sp
381 .sp
382 .LP
383 or

```

```

385 .sp
386 .in +2
387 .nf
388 example% \fB/usr/bin/grep -v \fR
389 .fi
390 .in -2
391 .sp

393 .LP
394 \fBExample 3 \fRFinding Lines Containing Strings
395 .sp
396 .LP
397 All of the following commands print all lines containing strings \fBabc\fR or
398 \fBdef\fR or both:

400 .sp
401 .in +2
402 .nf
403 example% \fB/usr/xpg4/bin/grep 'abc
404 def'\fR
405 example% \fB/usr/xpg4/bin/grep -e 'abc
406 def'\fR
407 example% \fB/usr/xpg4/bin/grep -e 'abc' -e 'def'\fR
408 example% \fB/usr/xpg4/bin/grep -E 'abc|def'\fR
409 example% \fB/usr/xpg4/bin/grep -E -e 'abc|def'\fR
410 example% \fB/usr/xpg4/bin/grep -E -e 'abc' -e 'def'\fR
411 example% \fB/usr/xpg4/bin/grep -E 'abc
412 def'\fR
413 example% \fB/usr/xpg4/bin/grep -E -e 'abc
414 def'\fR
415 example% \fB/usr/xpg4/bin/grep -F -e 'abc' -e 'def'\fR
416 example% \fB/usr/xpg4/bin/grep -F 'abc
417 def'\fR
418 example% \fB/usr/xpg4/bin/grep -F -e 'abc
419 def'\fR
420 .fi
421 .in -2
422 .sp

424 .LP
425 \fBExample 4 \fRFinding Lines with Matching Strings
426 .sp
427 .LP
428 Both of the following commands print all lines matching exactly \fBabc\fR or
429 \fBdef\fR:

431 .sp
432 .in +2
433 .nf
434 example% \fB/usr/xpg4/bin/grep -E '^abc$ ^def$'\fR
435 example% \fB/usr/xpg4/bin/grep -F -x 'abc def'\fR
436 .fi
437 .in -2
438 .sp

440 .SH ENVIRONMENT VARIABLES
441 .sp
442 .LP
443 See \fBenviron\fR(5) for descriptions of the following environment variables
444 that affect the execution of \fBgrep\fR: \fBBLANG\fR, \fBLC_ALL\fR,
445 \fBLC_COLLATE\fR, \fBLC_CTYPE\fR, \fBLC_MESSAGES\fR, and \fBLC_PATH\fR.
446 .SH EXIT STATUS
447 .sp
448 .LP
449 The following exit values are returned:

```

```

450 .sp
451 .ne 2
452 .na
453 \fB\b0\fR\fR
454 .ad
455 .RS 5n
456 One or more matches were found.
457 .RE

459 .sp
460 .ne 2
461 .na
462 \fB\b1\fR\fR
463 .ad
464 .RS 5n
465 No matches were found.
466 .RE

468 .sp
469 .ne 2
470 .na
471 \fB\b2\fR\fR
472 .ad
473 .RS 5n
474 Syntax errors or inaccessible files (even if matches were found).
475 .RE

477 .SH ATTRIBUTES
478 .sp
479 .LP
480 See \fBattributes\fR(5) for descriptions of the following attributes:
481 .SS "/usr/bin/grep"
482 .sp

484 .sp
485 .TS
486 box;
487 c | c
488 l | l .
489 ATTRIBUTE TYPE ATTRIBUTE VALUE
490 _
491 CSI Not Enabled
492 .TE

494 .SS "/usr/xpg4/bin/grep"
495 .sp

497 .sp
498 .TS
499 box;
500 c | c
501 l | l .
502 ATTRIBUTE TYPE ATTRIBUTE VALUE
503 _
504 CSI Enabled
505 _
506 Interface Stability Committed
507 _
508 Standard See \fBstandards\fR(5).
509 .TE

511 .SH SEE ALSO
512 .sp
513 .LP
514 \fBegrep\fR(1), \fBfgrep\fR(1), \fBsed\fR(1), \fBsh\fR(1), \fBattributes\fR(5),
515 \fBenviron\fR(5), \fBlargefile\fR(5), \fBregex\fR(5), \fBregexp\fR(5),

```



```
516 \fBstandards\fR(5)
517 .SH NOTES
518 .SS "/usr/bin/grep"
519 .sp
520 .LP
521 Lines are limited only by the size of the available virtual memory. If there is
522 a line with embedded nulls, \fBgrep\fR only matches up to the first null. If
523 the line matches, the entire line is printed.
524 .SS "/usr/xpg4/bin/grep"
525 .sp
526 .LP
527 The results are unspecified if input files contain lines longer than
528 \fBLINE_MAX\fR bytes or contain binary data. \fBLINE_MAX\fR is defined in
529 \fB/usr/include/limits.h\fR.
```