```
**********************************************************
   26107 Thu May 30 19:29:52 2013
new/usr/src/cmd/egrep/egrep.y
3737 grep does not support -H option
**********************************************************
```

```
%{
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
%}
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

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

/*
 * Copyright 2013 Damian Bogel. All rights reserved.
 */
%{
#pragma ident   "%Z%%M% %I%     %E% SMI"
%}

/*
 * egrep -- print lines containing (or not containing) a regular expression
 *
 *      status returns:
 *              0 - ok, and some matches
 *              1 - ok, but no matches
 *              2 - some error; matches irrelevant
 */
%token CHAR MCHAR DOT MDOT CCL NCCL MCCL NMCCL OR CAT STAR PLUS QUEST
%left OR
%left CHAR MCHAR DOT CCL NCCL MCCL NMCCL '('
%left CAT
%left STAR PLUS QUEST

%{
#include <stdio.h>
#include <ctype.h>
#include <memory.h>
#include <wchar.h>
#include <wctype.h>
```



```
#include <widec.h>
#include <stdlib.h>
#include <limits.h>
#include <locale.h>

#define STDIN_FILENAME gettext("(standard input)")

#endif /* ! codereview */
#define BLKSIZE 512     /* size of reported disk blocks */
#define EBUFSIZ 8192
#define MAXLIN 350
#define NCHARS 256
#define MAXPOS 4000
#define NSTATES 64
#define FINAL -1
#define RIGHT '\n'      /* serves as record separator and as $ */
#define LEFT '\n'       /* beginning of line */
int gotofn[NSTATES][NCHARS];
int state[NSTATES];
int out[NSTATES];
int line  = 1;
int *name;
int *left;
int *right;
int *parent;
int *foll;
int *positions;
char *chars;
wchar_t *lower;
wchar_t *upper;
int maxlin, maxclin, maxwclin, maxpos;
int nxtpos = 0;
int inxtpos;
int nxtchar = 0;
int *tmpstat;
int *initstat;
int istat;
int nstate = 1;
int xstate;
int count;
int icount;
char *input;


wchar_t lyylval;
wchar_t nextch();
wchar_t maxmin();
int compare();
void overflo();

char reinit = 0;

long long lnum;
int     bflag;
int     cflag;
int     eflag;
int     fflag;
int     Hflag;
#endif /* ! codereview */
int     hflag;
int     iflag;
int     lflag;
int     nflag;
int     qflag;
int     sflag;
int     vflag;
```

```
124 int     nfile;
125 long long blkno;
126 long long tln;
127 int     nsucc;
128 int     badbotch;
129 extern  char *optarg;
130 extern  int optind;

132 int     f;
133 FILE    *expfile;
134 %}

136 %%
137 s:      t
138                 {
139                   unary(FINAL, $1);
140                   line--;
141                 }
142         ;
143 t:      b r
144                 { $$ = node(CAT, $1, $2); }
145         | OR b r OR
146                 { $$ = node(CAT, $2, $3); }
147         | OR b r
148                 { $$ = node(CAT, $2, $3); }
149         | b r OR
150                 { $$ = node(CAT, $1, $2); }
151         ;
152 b:
153                 { /* if(multibyte)
154                         $$ = mdotenter();
155                   else */
156                         $$ = enter(DOT);
157                   $$ = unary(STAR, $$);
158                 }
159         ;
160 r:      CHAR
161                 { $$ = iflag && isalpha($1) ?
162                 node(OR, enter(tolower($1)), enter(toupper($1))) : enter($1); }
163         | MCHAR
164                 { $$ = (iflag && iswalpha(lyylval)) ?
165                 node(OR, mchar(towlower(lyylval)), mchar(towupper(lyylval))) :
166                 mchar(lyylval); }
167         | DOT
168                 { if(multibyte)
169                         $$ = mdotenter();
170                   else
171                         $$ = enter(DOT);
172                 }
173         | CCL
174                 { $$ = cclenter(CCL); }
175         | NCCL
176                 { $$ = cclenter(NCCL); }
177         | MCCL
178                 { $$ = ccl(CCL); }
179         | NMCCL
180                 { $$ = ccl(NCCL); }
181         ;

183 r:      r OR r
184                 { $$ = node(OR, $1, $3); }
185         | r r %prec CAT
186                 { $$ = node(CAT, $1, $2); }
187         | r STAR
188                 { $$ = unary(STAR, $1); }
189         | r PLUS
```



```
190                 { $$ = unary(PLUS, $1); }
191         | r QUEST
192                 { $$ = unary(QUEST, $1); }
193         | '(' r ')'
194                 { $$ = $2; }
195         | error
196         ;

198 %%
199 void    add(int *, int);
200 void    clearg(void);
201 void    execute(char *);
202 void    follow(int);
203 int     mgetc(void);
204 void    synerror(void);


207 void
208 yyerror(char *s)
209 {
210         fprintf(stderr, "egrep: %s\n", s);
211         exit(2);
212 }
```

*unchanged_portion_omitted_*

```
652 #define USAGE "[ -bchilnsv ] [ -e exp ] [ -f file ] [ strings ] [ file ] ..."

654 int
655 main(int argc, char **argv)
656 {
657         char c;
658         char nl = '\n';
659         int errflag = 0;
660
661         (void)setlocale(LC_ALL, "");

663 #if !defined(TEXT_DOMAIN)               /* Should be defined by cc -D */
664         #define TEXT_DOMAIN "SYS_TEST"  /* Use this only if it weren't. */
665 #endif
666         (void) textdomain(TEXT_DOMAIN);

668         while((c = getopt(argc, argv, "ybcie:f:Hhlnvs")) != -1)
610         while((c = getopt(argc, argv, "ybcie:f:hlnvs")) != -1)
669                 switch(c) {

671                 case 'b':
672                         bflag++;
673                         continue;

675                 case 'c':
676                         cflag++;
677                         continue;

679                 case 'e':
680                         eflag++;
681                         input = optarg;
682                         continue;

684                 case 'f':
685                         fflag++;
686                         expfile = fopen(optarg, "r");
687                         if(expfile == NULL) {
688                                 fprintf(stderr,
689                                   gettext("egrep: can't open %s\n"), optarg);
690                                 exit(2);
691                         }
```

```
692                         continue;

694                 case 'H':
695                         if (!lflag) /* H is excluded by l as in GNU grep */
696                                 Hflag++;
697                         hflag = 0; /* H excludes h */
698                         continue;

700 #endif /* ! codereview */
701                 case 'h':
702                         hflag++;
703                         Hflag = 0; /* h excludes H */
704 #endif /* ! codereview */
705                         continue;

707                 case 'y':
708                 case 'i':
709                         iflag++;
710                         continue;

712                 case 'l':
713                         lflag++;
714                         Hflag = 0; /* l excludes H */
715 #endif /* ! codereview */
716                         continue;

718                 case 'n':
719                         nflag++;
720                         continue;

722                 case 'q':
723                 case 's': /* Solaris: legacy option */
724                         qflag++;
636                 case 's':
637                         sflag++;
725                         continue;

727                 case 'v':
728                         vflag++;
729                         continue;

731                 case '?':
732                         errflag++;
733                 }
734         if (errflag || ((argc <= 0) && !fflag && !eflag)) {
735                 fprintf(stderr, gettext("usage: egrep %s\n"), gettext(USAGE));
736                 exit(2);
737         }
738         if(!eflag && !fflag) {
739                 input = argv[optind];
740                 optind++;
741         }

743         argc -= optind;
744         argv = &argv[optind];
745
746         /* allocate initial space for arrays */
747         if((name = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
748                 overflo();
749         if((left = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
750                 overflo();
751         if((right = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
752                 overflo();
753         if((parent = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
754                 overflo();
755         if((foll = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
```



```
756                 overflo();
757         if((tmpstat = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
758                 overflo();
759         if((initstat = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
760                 overflo();
761         if((chars = (char *)malloc(MAXLIN)) == (char *)0)
762                 overflo();
763         if((lower = (wchar_t *)malloc(MAXLIN*sizeof(wchar_t))) == (wchar_t *)0)
764                 overflo();
765         if((upper = (wchar_t *)malloc(MAXLIN*sizeof(wchar_t))) == (wchar_t *)0)
766                 overflo();
767         if((positions = (int *)malloc(MAXPOS*sizeof(int))) == (int *)0)
768                 overflo();
769         maxlin = MAXLIN;
770         maxclin = MAXLIN;
771         maxwclin = MAXLIN;
772         maxpos = MAXPOS;
773
774         yyparse();

776         cfoll(line-1);
777         cgotofn();
778         nfile = argc;
779         if (argc<=0) {
780                 execute(0);
781         }
782         else while (--argc >= 0) {
783                 if (reinit == 1) clearg();
784                 execute(*argv++);
785         }
786         return (badbotch ? 2 : nsucc==0);
787 }

789 void
790 execute(char *file)
791 {
792         char *p;
793         int cstat;
794         wchar_t c;
795         int t;
796         long count;
797         long count1, count2;
798         long nchars;
799         int succ;
800         char *ptr, *ptrend, *lastptr;
801         char *buf;
802         long lBufSiz;
803         FILE *f;
804         int nlflag;

806         lBufSiz = EBUFSIZ;
807         if ((buf = malloc (lBufSiz + EBUFSIZ)) == NULL) {
808                 exit (2); /* out of memory - BAIL */
809         }

811         if (file) {
812                 if ((f = fopen(file, "r")) == NULL) {
813                         fprintf(stderr,
814                                 gettext("egrep: can't open %s\n"), file);
815                         badbotch=1;
816                         return;
817                 }
818         } else {
732                 file = "<stdin>";
819                 f = stdin;
820                 file = STDIN_FILENAME;
```

```
#endif /* ! codereview */
        }
        lnum = 1;
        tln = 0;
        if((count = read(fileno(f), buf, EBUFSIZ)) <= 0) {
                fclose(f);

                if (cflag && !qflag) {
                        if (Hflag || (nfile > 1 && !hflag))
                if (cflag) {
                        if (nfile>1 && !hflag)
                                fprintf(stdout, "%s:", file);
                        fprintf(stdout, "%lld\n", tln);
                }
                return;
        }

        blkno = count;
        ptr = buf;
        for(;;) {
                if((ptrend = memchr(ptr, '\n', buf + count - ptr)) == NULL) {
                        /*
                                move the unused partial record to the head of th
                        */
                        if (ptr > buf) {
                                count = buf + count - ptr;
                                memmove (buf, ptr, count);
                                ptr = buf;
                        }

                        /*
                                Get a bigger buffer if this one is full
                        */
                        if(count > lBufSiz) {
                                /*
                                        expand the buffer
                                */
                                lBufSiz += EBUFSIZ;
                                if ((buf = realloc (buf, lBufSiz + EBUFSIZ)) ==
                                        exit (2); /* out of memory - BAIL */
                                }

                                ptr = buf;
                        }

                        p = buf + count;
                        if((count1 = read(fileno(f), p, EBUFSIZ)) > 0) {
                                count += count1;
                                blkno += count1;
                                continue;
                        }
                        ptrend = ptr + count;
                        nlflag = 0;
                } else
                        nlflag = 1;
                *ptrend = '\n';
                p = ptr;
                lastptr = ptr;
                cstat = istat;
                succ = 0;
                for(;;) {
                        if(out[cstat]) {
                                if(multibyte && p > ptr) {
                                        wchar_t wchar;
                                        int length;
                                        char *endptr = p;
```



```
                                        p = lastptr;
                                        while(p < endptr) {
                                                length = mbtowc(&wchar, p, MB_LE
                                                if(length <= 1)
                                                        p++;
                                                else
                                                        p += length;
                                        }
                                        if(p == endptr) {
                                                succ = !vflag;
                                                break;
                                        }
                                        cstat = 1;
                                        length = mbtowc(&wchar, lastptr, MB_LEN_
                                        if(length <= 1)
                                                lastptr++;
                                        else
                                                lastptr += length;
                                        p = lastptr;
                                        continue;
                                }
                                succ = !vflag;
                                break;
                        }
                        c = (unsigned char)*p++;
                        if ((t = gotofn[cstat][c]) == 0)
                                cstat = nxtst(cstat, c);
                        else
                                cstat = t;
                        if(c == RIGHT) {
                                if(out[cstat]) {
                                        succ = !vflag;
                                        break;
                                }
                                succ = vflag;
                                break;
                        }
                }
                if (succ) {
                if(succ) {
                        nsucc = 1;
                        if (lflag || qflag) {
                                if (!qflag)
                                        (void) printf("%s\n", file);
                        if (cflag) tln++;
                        else if (sflag)
                                ;       /* ugh */
                        else if (lflag) {
                                printf("%s\n", file);
                                fclose(f);
                                return;
                        }
                        if (cflag) {
                                tln++;
                        } else {
                                if (Hflag || (nfile > 1 && !hflag))
                                        printf("%s:", file);
                        else {
                                if (nfile > 1 && !hflag)
                                        printf(gettext("%s:"), file);
                                if (bflag) {
                                        nchars = blkno - (buf + count - ptrend)
                                        if(nlflag)
                                                nchars++;
                                        printf("%lld:", nchars/BLKSIZE);
                                }
```

```
942                                 if (nflag)
943                                         printf("%lld:", lnum);
944                                 if(nlflag)
945                                         nchars = ptrend - ptr + 1;
946                                 else
947                                         nchars = ptrend - ptr;
948                                 fwrite(ptr, (size_t)1, (size_t)nchars, stdout);
949                         }
950                 }
951                 if(!nlflag)
952                         break;
953                 ptr = ptrend + 1;
954                 if(ptr >= buf + count) {
955                         ptr = buf;
956                         if((count = read(fileno(f), buf, EBUFSIZ)) <= 0)
957                                 break;
958                         blkno += count;
959                 }
960                 lnum++;
961                 if (reinit == 1)
962                         clearg();
963         }
964         fclose(f);
965         if (cflag && !qflag) {
966                 if (Hflag || (nfile > 1 && !hflag))
967                         printf("%s:", file);
871         if (cflag) {
872                 if (nfile > 1 && !hflag)
873                         printf(gettext("%s:"), file);
968                 printf("%lld\n", tln);
969         }
970 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   14443 Thu May 30 19:29:52 2013
new/usr/src/cmd/fgrep/fgrep.c
3737 grep does not support -H option
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

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

/*
 * Copyright 2013 Damian Bogel. All rights reserved.
 */
#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * fgrep -- print all lines containing any of a set of keywords
 *
 *      status returns:
 *              0 - ok, and some matches
 *              1 - ok, but no matches
 *              2 - some error
 */

#include <stdio.h>
#include <ctype.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#include <libintl.h>
#include <euc.h>
#include <sys/stat.h>
#include <fcntl.h>

#include <getwidth.h>

eucwidth_t WW;
#define WIDTH1  WW._eucw1
```



```
#define WIDTH2  WW._eucw2
#define WIDTH3  WW._eucw3
#define MULTI_BYTE      WW._multibyte
#define GETONE(lc, p) \
        cw = ISASCII(lc = (unsigned char)*p++) ? 1 :     \
                (ISSET2(lc) ? WIDTH2 :                          \
                (ISSET3(lc) ? WIDTH3 : WIDTH1));                \
        if (--cw > --ccount) {                                  \
                cw -= ccount;                                   \
                while (ccount--)                                \
                        lc = (lc << 7) | ((*p++) & 0177);       \
                        if (p >= &buf[fw_lBufsiz + BUFSIZ]) {   \
                        if (nlp == buf) {                       \
                                /* Increase the buffer size */      \
                                fw_lBufsiz += BUFSIZ;                   \
                                if ((buf = realloc(buf,                 \
                                        fw_lBufsiz + BUFSIZ)) == NULL) { \
                                        exit(2); /* out of memory */    \
                                }                                       \
                                nlp = buf;                              \
                                p = &buf[fw_lBufsiz];                   \
                        } else {                                        \
                                /* shift the buffer contents down */ \
                                (void) memmove(buf, nlp,                \
                                        &buf[fw_lBufsiz + BUFSIZ] - nlp);\
                                p -= nlp - buf;                         \
                                nlp = buf;                              \
                        }                                               \
                }                                                       \
                if (p > &buf[fw_lBufsiz]) {                             \
                        if ((ccount = fread(p, sizeof (char),    \
                            &buf[fw_lBufsiz + BUFSIZ] - p, fptr))\
                                <= 0) break;                            \
                } else if ((ccount = fread(p,                           \
                        sizeof (char),  BUFSIZ, fptr)) <= 0)    \
                        break;                                          \
                blkno += (long long)ccount;                             \
        }                                                       \
        ccount -= cw;                                           \
        while (cw--)                                            \
                lc = (lc << 7) | ((*p++) & 0177)

/*
 * The same() macro and letter() function were inserted to allow for
 * the -i option work for the multi-byte environment.
 */
wchar_t letter();
#define same(a, b) \
        (a == b || iflag && (!MULTI_BYTE || ISASCII(a)) && (a ^ b) == ' ' && \
        letter(a) == letter(b))

#define STDIN_FILENAME gettext("(standard input)")
#endif /* ! codereview */

#define QSIZE 400
struct words {
        wchar_t inp;
        char    out;
        struct  words *nst;
        struct  words *link;
        struct  words *fail;
} *w = NULL, *smax, *q;

FILE *fptr;
long long lnum;
int     bflag, cflag, lflag, fflag, nflag, vflag, xflag, eflag, qflag;
```

```
127 int     Hflag, hflag, iflag;
110 int     bflag, cflag, lflag, fflag, nflag, vflag, xflag, eflag, sflag;
111 int     hflag, iflag;
128 int     retcode = 0;
129 int     nfile;
130 long long blkno;
131 int     nsucc;
132 long long tln;
133 FILE    *wordf;
134 char    *argptr;
135 off_t input_size = 0;

137 void    execute(char *);
138 void    cgotofn(void);
139 void    overflo(void);
140 void    cfail(void);

142 static long fw_lBufsiz = 0;

144 int
145 main(int argc, char **argv)
146 {
147         int c;
148         int errflg = 0;
149         struct stat file_stat;

151         (void) setlocale(LC_ALL, "");
152 #if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
153 #define TEXT_DOMAIN "SYS_TEST"  /* Use this only if it weren't */
154 #endif
155         (void) textdomain(TEXT_DOMAIN);

157         while ((c = getopt(argc, argv, "Hhybcie:f:lnvxqs")) != EOF)
141         while ((c = getopt(argc, argv, "hybcie:f:lnvxs")) != EOF)
158                 switch (c) {

160                 case 'q':
161                 case 's': /* Solaris: legacy option */
162                         qflag++;
163                         continue;
164                 case 'H':
165                         Hflag++;
166                         hflag = 0;
144                 case 's':
145                         sflag++;
167                         continue;
168                 case 'h':
169                         hflag++;
170                         Hflag = 0;
171 #endif /* ! codereview */
172                         continue;
173                 case 'b':
174                         bflag++;
175                         continue;

177                 case 'i':
178                 case 'y':
179                         iflag++;
180                         continue;

182                 case 'c':
183                         cflag++;
184                         continue;

186                 case 'e':
187                         eflag++;
```



```
188                         argptr = optarg;
189                         input_size = strlen(argptr);
190                         continue;

192                 case 'f':
193                         fflag++;
194                         wordf = fopen(optarg, "r");
195                         if (wordf == NULL) {
196                                 (void) fprintf(stderr,
197                                         gettext("fgrep: can't open %s\n"),
198                                         optarg);
199                                 exit(2);
200                         }

202                         if (fstat(fileno(wordf), &file_stat) == 0) {
203                             input_size = file_stat.st_size;
204                         } else {
205                                 (void) fprintf(stderr,
206                                         gettext("fgrep: can't fstat %s\n"),
207                                         optarg);
208                                 exit(2);
209                         }

211                         continue;

213                 case 'l':
214                         lflag++;
215                         continue;

217                 case 'n':
218                         nflag++;
219                         continue;

221                 case 'v':
222                         vflag++;
223                         continue;

225                 case 'x':
226                         xflag++;
227                         continue;

229                 case '?':
230                         errflg++;
231         }

233         argc -= optind;
234         if (errflg || ((argc <= 0) && !fflag && !eflag)) {
235                 (void) printf(gettext("usage: fgrep [ -bcHhilnqsvx ] "
149                 (void) printf(gettext("usage: fgrep [ -bchilnsvx ] "
236                         "[ -e exp ] [ -f file ] [ strings ] [ file ] ...\n"));
237                 exit(2);
238         }
239         if (!eflag && !fflag) {
240                 argptr = argv[optind];
241                 input_size = strlen(argptr);
242                 input_size++;
243                 optind++;
244                 argc--;
245         }

247 /*
248  * Normally we need one struct words for each letter in the pattern
249  * plus one terminating struct words with outp = 1, but when -x option
250  * is specified we require one more struct words for '\n' character so we
251  * calculate the input_size as below. We add extra 1 because
252  * (input_size/2) rounds off odd numbers
```

```
 */

        if (xflag) {
                input_size = input_size + (input_size/2) + 1;
        }

        input_size++;

        w = (struct words *)calloc(input_size, sizeof (struct words));
        if (w == NULL) {
                (void) fprintf(stderr,
                        gettext("fgrep: could not allocate "
                                "memory for wordlist\n"));
                exit(2);
        }

        getwidth(&WW);
        if ((WIDTH1 == 0) && (WIDTH2 == 0) &&
                (WIDTH3 == 0)) {
                /*
                 * If non EUC-based locale,
                 * assume WIDTH1 is 1.
                 */
                WIDTH1 = 1;
        }
        WIDTH2++;
        WIDTH3++;

        cgotofn();
        cfail();
        nfile = argc;
        argv = &argv[optind];
        if (argc <= 0) {
                execute((char *)NULL);
        } else
                while (--argc >= 0) {
                        execute(*argv);
                        argv++;
                }

        if (w != NULL) {
                free(w);
        }

        return (retcode != 0 ? retcode : nsucc == 0);
}

void
execute(char *file)
{
        char *p;
        struct words *c;
        int ccount;
        static char *buf = NULL;
        int failed;
        char *nlp;
        wchar_t lc;
        int cw;

        if (buf == NULL) {
                fw_lBufsiz = BUFSIZ;
                if ((buf = malloc(fw_lBufsiz + BUFSIZ)) == NULL) {
                        exit(2); /* out of memory */
                }
        }
```



```
        if (file) {
                if ((fptr = fopen(file, "r")) == NULL) {
                        (void) fprintf(stderr,
                                gettext("fgrep: can't open %s\n"), file);
                        retcode = 2;
                        return;
                }
        } else {
                file = "<stdin>";
                fptr = stdin;
                file = STDIN_FILENAME;
#endif /* ! codereview */
        }
        ccount = 0;
        failed = 0;
        lnum = 1;
        tln = 0;
        blkno = 0;
        p = buf;
        nlp = p;
        c = w;
        for (;;) {
                if (c == 0)
                        break;
                if (ccount <= 0) {
                        if (p >= &buf[fw_lBufsiz + BUFSIZ]) {
                                if (nlp == buf) {
                                        /* increase the buffer size */
                                        fw_lBufsiz += BUFSIZ;
                                        if ((buf = realloc(buf,
                                                fw_lBufsiz + BUFSIZ)) == NULL) {
                                                exit(2); /* out of memory */
                                        }
                                        nlp = buf;
                                        p = &buf[fw_lBufsiz];
                                } else {
                                        /* shift the buffer down */
                                        (void) memmove(buf, nlp,
                                                &buf[fw_lBufsiz + BUFSIZ]
                                                - nlp);
                                        p -= nlp - buf;
                                        nlp = buf;
                                }

                        }
                        if (p > &buf[fw_lBufsiz]) {
                                if ((ccount = fread(p, sizeof (char),
                                        &buf[fw_lBufsiz + BUFSIZ] - p, fptr))
                                        <= 0)
                                        break;
                        } else if ((ccount = fread(p, sizeof (char),
                                BUFSIZ, fptr)) <= 0)
                                break;
                        blkno += (long long)ccount;
                }
                GETONE(lc, p);
nstate:
                if (same(c->inp, lc)) {
                        c = c->nst;
                } else if (c->link != 0) {
                        c = c->link;
                        goto nstate;
                } else {
                        c = c->fail;
                        failed = 1;
                        if (c == 0) {
```

```
384                                 c = w;
385 istate:
386                                 if (same(c->inp, lc)) {
387                                         c = c->nst;
388                                 } else if (c->link != 0) {
389                                         c = c->link;
390                                         goto istate;
391                                 }
392                         } else
393                                 goto nstate;
394                 }

396                 if (c == 0)
397                         break;

399                 if (c->out) {
400                         while (lc != '\n') {
401                                 if (ccount <= 0) {
402 if (p == &buf[fw_lBufsiz + BUFSIZ]) {
403         if (nlp == buf) {
404                 /* increase buffer size */
405                 fw_lBufsiz += BUFSIZ;
406                 if ((buf = realloc(buf, fw_lBufsiz + BUFSIZ)) == NULL) {
407                         exit(2); /* out of memory */
408                 }
409                 nlp = buf;
410                 p = &buf[fw_lBufsiz];
411         } else {
412                 /* shift buffer down */
413                 (void) memmove(buf, nlp, &buf[fw_lBufsiz + BUFSIZ] - nlp);
414                 p -= nlp - buf;
415                 nlp = buf;
416         }
417 }
418 if (p > &buf[fw_lBufsiz]) {
419         if ((ccount = fread(p, sizeof (char),
420                 &buf[fw_lBufsiz + BUFSIZ] - p, fptr)) <= 0) break;
421         } else if ((ccount = fread(p, sizeof (char), BUFSIZ,
422                 fptr)) <= 0) break;
423                 blkno += (long long)ccount;
424         }
425         GETONE(lc, p);
426 }
427                         if ((vflag && (failed == 0 || xflag == 0)) ||
428                                 (vflag == 0 && xflag && failed))
429                                 goto nomatch;
430 succeed:
431                         nsucc = 1;
432                         if (lflag || qflag) {
433                                 if (!qflag)
243                         if (cflag)
244                                 tln++;
245                         else if (lflag && !sflag) {
434                                         (void) printf("%s\n", file);
435                                 (void) fclose(fptr);
436                                 return;
437                         }
438                         if (cflag) {
439                                 tln++;
440                         } else {
441                                 if (Hflag || (nfile > 1 && !hflag))
249                         } else if (!sflag) {
250                                 if (nfile > 1 && !hflag)
442                                         (void) printf("%s:", file);
443                                 if (bflag)
444                                         (void) printf("%lld:",
```



```
445                                                 (blkno - (long long)(ccount-1))
446                                                 / BUFSIZ);
447                                 if (nflag)
448                                         (void) printf("%lld:", lnum);
449                                 if (p <= nlp) {
450                                         while (nlp < &buf[fw_lBufsiz + BUFSIZ])
451                                                 (void) putchar(*nlp++);
452                                         nlp = buf;
453                                 }
454                                 while (nlp < p)
455                                         (void) putchar(*nlp++);
456                         }
457 nomatch:
458                         lnum++;
459                         nlp = p;
460                         c = w;
461                         failed = 0;
462                         continue;
463                 }
464                 if (lc == '\n')
465                         if (vflag)
466                                 goto succeed;
467                         else {
468                                 lnum++;
469                                 nlp = p;
470                                 c = w;
471                                 failed = 0;
472                         }
473         }
474         (void) fclose(fptr);
475         if (cflag && !qflag) {
476                 if (Hflag || (nfile > 1 && !hflag))
284         if (cflag) {
285                 if ((nfile > 1) && !hflag)
477                         (void) printf("%s:", file);
478                 (void) printf("%lld\n", tln);
479         }
480 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   10567 Thu May 30 19:29:52 2013
new/usr/src/cmd/grep/grep.c
3737 grep does not support -H option
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

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

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

/*
 * Copyright 2013 Damian Bogel. All rights reserved.
 */

/*
#endif /* ! codereview */
 * grep -- print lines matching (or not matching) a pattern
 *
 *      status returns:
 *              0 - ok, and some matches
 *              1 - ok, but no matches
 *              2 - some error
 */

#include <sys/types.h>

#include <ctype.h>
#include <fcntl.h>
#include <locale.h>
#include <memory.h>
#include <regexpr.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <ftw.h>
#include <limits.h>
```



```
#include <sys/param.h>

static const char *errstr[] = {
        "Range endpoint too large.",
        "Bad number.",
        "‘‘\\digit’’ out of range.",
        "No remembered search string.",
        "\\( \\) imbalance.",
        "Too many \\(.",
        "More than 2 numbers given in \\{ \\}.",
        "} expected after \\.",
        "First number exceeds second in \\{ \\}.",
        "[ ] imbalance.",
        "Regular expression overflow.",
        "Illegal byte sequence.",
        "Unknown regexp error code!!",
        NULL
};

#define STDIN_FILENAME  gettext("(standard input)")

#endif /* ! codereview */
#define errmsg(msg, arg)        (void) fprintf(stderr, gettext(msg), arg)
#define BLKSIZE 512
#define GBUFSIZ 8192
#define MAX_DEPTH       1000

static int      temp;
static long long        lnum;
static char     *linebuf;
static char     *prntbuf = NULL;
static long     fw_lPrntBufLen = 0;
static int      nflag;
static int      bflag;
static int      lflag;
static int      cflag;
static int      rflag;
static int      Rflag;
static int      vflag;
static int      sflag;
static int      iflag;
static int      wflag;
static int      hflag;
static int      Hflag;
#endif /* ! codereview */
static int      qflag;
static int      errflg;
static int      nfile;
static long long        tln;
static int      nsucc;
static int      outfn = 0;
static int      nlflag;
static char     *ptr, *ptrend;
static char     *expbuf;

static void     execute(const char *, int);
static void     regerr(int);
static void     prepare(const char *);
static int      recursive(const char *, const struct stat *, int, struct FTW *);
static int      succeed(const char *);

int
main(int argc, char **argv)
{
        int     c;
        char    *arg;
```

```
128         extern int      optind;

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

136         while ((c = getopt(argc, argv, "hHqblcnRrsviyw")) != -1)
 36         while ((c = getopt(argc, argv, "hqblcnRrsviyw")) != -1)
137                 switch (c) {
138                 /* based on options order h or H is set as in GNU grep */
139 #endif /* ! codereview */
140                 case 'h':
141                         hflag++;
142                         Hflag = 0; /* h excludes H */
143                         break;
144                 case 'H':
145                         if (!lflag) /* H is excluded by l */
146                                 Hflag++;
147                         hflag = 0; /* H excludes h */
148 #endif /* ! codereview */
149                         break;
150                 case 'q':       /* POSIX: quiet: status only */
151                         qflag++;
152                         break;
153                 case 'v':
154                         vflag++;
155                         break;
156                 case 'c':
157                         cflag++;
158                         break;
159                 case 'n':
160                         nflag++;
161                         break;
162                 case 'R':
163                         Rflag++;
164                         /* FALLTHROUGH */
165                 case 'r':
166                         rflag++;
167                         break;
168                 case 'b':
169                         bflag++;
170                         break;
171                 case 's':
172                         sflag++;
173                         break;
174                 case 'l':
175                         lflag++;
176                         Hflag = 0; /* l excludes H */
177 #endif /* ! codereview */
178                         break;
179                 case 'y':
180                 case 'i':
181                         iflag++;
182                         break;
183                 case 'w':
184                         wflag++;
185                         break;
186                 case '?':
187                         errflg++;
188                 }

190         if (errflg || (optind >= argc)) {
191                 errmsg("Usage: grep [-c|-l|-q] [-r|-R] -hHbnsviw "
 38                 errmsg("Usage: grep [-c|-l|-q] [-r|-R] -hbnsviw "
```



```
192                     "pattern file . . .\n",
193                     (char *)NULL);
194                 exit(2);
195         }

197         argv = &argv[optind];
198         argc -= optind;
199         nfile = argc - 1;

201         if (strrchr(*argv, '\n') != NULL)
202                 regerr(41);

204         if (iflag) {
205                 for (arg = *argv; *arg != NULL; ++arg)
206                         *arg = (char)tolower((int)((unsigned char)*arg));
207         }

209         if (wflag) {
210                 unsigned int    wordlen;
211                 char            *wordbuf;

213                 wordlen = strlen(*argv) + 5; /* '\\' '<' *argv '\\' '>' '\0' */
214                 if ((wordbuf = malloc(wordlen)) == NULL) {
215                         errmsg("grep: Out of memory for word\n", (char *)NULL);
216                         exit(2);
217                 }

219                 (void) strcpy(wordbuf, "\\<");
220                 (void) strcat(wordbuf, *argv);
221                 (void) strcat(wordbuf, "\\>");
222                 *argv = wordbuf;
223         }

225         expbuf = compile(*argv, (char *)0, (char *)0);
226         if (regerrno)
227                 regerr(regerrno);

229         if (--argc == 0)
230                 execute(NULL, 0);
231         else
232                 while (argc-- > 0)
233                         prepare(*++argv);

235         return (nsucc == 2 ? 2 : (nsucc == 0 ? 1 : 0));
236 }
_____unchanged_portion_omitted_

302 static void
303 execute(const char *file, int base)
304 {
305         char    *lbuf, *p;
306         long    count;
307         long    offset = 0;
308         char    *next_ptr = NULL;
309         long    next_count = 0;

311         tln = 0;

313         if (prntbuf == NULL) {
314                 fw_lPrntBufLen = GBUFSIZ + 1;
315                 if ((prntbuf = malloc(fw_lPrntBufLen)) == NULL) {
316                         exit(2); /* out of memory - BAIL */
317                 }
318                 if ((linebuf = malloc(fw_lPrntBufLen)) == NULL) {
319                         exit(2); /* out of memory - BAIL */
320                 }
```

```
321         }

323         if (file == NULL) {
170         if (file == NULL)
324                 temp = 0;
325                 file = STDIN_FILENAME;
326         } else if ((temp = open(file + base, O_RDONLY)) == -1) {
172         else if ((temp = open(file + base, O_RDONLY)) == -1) {
327                 if (!sflag)
328                         errmsg("grep: can't open %s\n", file);
329                 nsucc = 2;
330                 return;
331         }

333         /* read in first block of bytes */
334         if ((count = read(temp, prntbuf, GBUFSIZ)) <= 0) {
335                 (void) close(temp);

337                 if (cflag && !qflag) {
338                         if (Hflag || (nfile > 1 && !hflag))
184                         if (nfile > 1 && !hflag && file)
339                                 (void) fprintf(stdout, "%s:", file);
340                         if (!rflag)
341                         (void) fprintf(stdout, "%lld\n", tln);
342                 }
343                 return;
344         }

346         lnum = 0;
347         ptr = prntbuf;
348         for (;;) {
349                 /* look for next newline */
350                 if ((ptrend = memchr(ptr + offset, '\n', count)) == NULL) {
351                         offset += count;

353                         /*
354                          * shift unused data to the beginning of the buffer
355                          */
356                         if (ptr > prntbuf) {
357                                 (void) memmove(prntbuf, ptr, offset);
358                                 ptr = prntbuf;
359                         }

361                         /*
362                          * re-allocate a larger buffer if this one is full
363                          */
364                         if (offset + GBUFSIZ > fw_lPrntBufLen) {
365                                 /*
366                                  * allocate a new buffer and preserve the
367                                  * contents...
368                                  */
369                                 fw_lPrntBufLen += GBUFSIZ;
370                                 if ((prntbuf = realloc(prntbuf,
371                                     fw_lPrntBufLen)) == NULL)
372                                         exit(2);

374                                 /*
375                                  * set up a bigger linebuffer (this is only used
376                                  * for case insensitive operations). Contents do
377                                  * not have to be preserved.
378                                  */
379                                 free(linebuf);
380                                 if ((linebuf = malloc(fw_lPrntBufLen)) == NULL)
381                                         exit(2);

383                                 ptr = prntbuf;
```



```
384                         }

386                         p = prntbuf + offset;
387                         if ((count = read(temp, p, GBUFSIZ)) > 0)
388                                 continue;

390                         if (offset == 0)
391                                 /* end of file already reached */
392                                 break;

394                         /* last line of file has no newline */
395                         ptrend = ptr + offset;
396                         nlflag = 0;
397                 } else {
398                         next_ptr = ptrend + 1;
399                         next_count = offset + count - (next_ptr - ptr);
400                         nlflag = 1;
401                 }
402                 lnum++;
403                 *ptrend = '\0';

405                 if (iflag) {
406                         /*
407                          * Make a lower case copy of the record
408                          */
409                         p = ptr;
410                         for (lbuf = linebuf; p < ptrend; )
411                                 *lbuf++ = (char)tolower((int)
412                                     (unsigned char)*p++);
413                         *lbuf = '\0';
414                         lbuf = linebuf;
415                 } else
416                         /*
417                          * Use record as is
418                          */
419                         lbuf = ptr;

421                 /* lflag only once */
422                 if ((step(lbuf, expbuf) ^ vflag) && succeed(file) == 1)
423                         break;

425                 if (!nlflag)
426                         break;

428                 ptr = next_ptr;
429                 count = next_count;
430                 offset = 0;
431         }
432         (void) close(temp);

434         if (cflag && !qflag) {
435                 if (Hflag || (!hflag && ((nfile > 1) ||
436                     (rflag && outfn))))
281                 if (!hflag && file && (nfile > 1 ||
282                     (rflag && outfn)))
437                         (void) fprintf(stdout, "%s:", file);
438                 (void) fprintf(stdout, "%lld\n", tln);
439         }
440 }

442 static int
443 succeed(const char *f)
444 {
445         int nchars;
446         nsucc = (nsucc == 2) ? 2 : 1;
```

```
294         if (f == NULL)
295                 f = "<stdin>";

448         if (qflag) {
449                 /* no need to continue */
450                 return (1);
451         }

453         if (cflag) {
454                 tln++;
455                 return (0);
456         }

458         if (lflag) {
459                 (void) fprintf(stdout, "%s\n", f);
460                 return (1);
461         }

463         if (Hflag || (!hflag && (nfile > 1 || (rflag && outfn)))) {
312         if (!hflag && (nfile > 1 || (rflag && outfn))) {
464                 /* print filename */
465                 (void) fprintf(stdout, "%s:", f);
466         }

468         if (bflag)
469                 /* print block number */
470                 (void) fprintf(stdout, "%lld:", (offset_t)
471                     ((lseek(temp, (off_t)0, SEEK_CUR) - 1) / BLKSIZE));

473         if (nflag)
474                 /* print line number */
475                 (void) fprintf(stdout, "%lld:", lnum);

477         if (nlflag) {
478                 /* newline at end of line */
479                 *ptrend = '\n';
480                 nchars = ptrend - ptr + 1;
481         } else {
482                 /* don't write sentinel \0 */
483                 nchars = ptrend - ptr;
484         }

486         (void) fwrite(ptr, 1, nchars, stdout);
487         return (0);
488 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   28372 Thu May 30 19:29:53 2013
new/usr/src/cmd/grep_xpg4/grep.c
3737 grep does not support -H option
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * Copyright 1985, 1992 by Mortice Kern Systems Inc.  All rights reserved.
 *
 */

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

/*
 * Copyright 2013 Damian Bogel. All rights reserved.
 */

#endif /* ! codereview */
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdarg.h>
#include <regex.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <locale.h>
#include <wchar.h>
#include <errno.h>
#include <unistd.h>
#include <wctype.h>
#include <ftw.h>
#include <sys/param.h>
```



```
#define STDIN_FILENAME gettext("(standard input)")

#endif /* ! codereview */
#define BSIZE           512             /* Size of block for -b */
#define BUFSIZE         8192            /* Input buffer size */
#define MAX_DEPTH       1000            /* how deep to recurse */

#define M_CSETSIZE      256             /* singlebyte chars */
static int      bmglen;                 /* length of BMG pattern */
static char     *bmgpat;                /* BMG pattern */
static int      bmgtab[M_CSETSIZE];     /* BMG delta1 table */

typedef struct  _PATTERN        {
        char    *pattern;               /* original pattern */
        wchar_t *wpattern;              /* wide, lowercased pattern */
        struct  _PATTERN        *next;
        regex_t re;                     /* compiled pattern */
} PATTERN;

static PATTERN  *patterns;
static char     errstr[128];            /* regerror string buffer */
static int      regflags = 0;           /* regcomp options */
static int      matched = 0;            /* return of the grep() */
static int      errors = 0;             /* count of errors */
static uchar_t  fgrep = 0;              /* Invoked as fgrep */
static uchar_t  egrep = 0;              /* Invoked as egrep */
static uchar_t  nvflag = 1;             /* Print matching lines */
static uchar_t  cflag;                  /* Count of matches */
static uchar_t  iflag;                  /* Case insensitve matching */
static uchar_t  Hflag;                  /* Precede lines by file name */
#endif /* ! codereview */
static uchar_t  hflag;                  /* Supress printing of filename */
static uchar_t  lflag;                  /* Print file names of matches */
static uchar_t  nflag;                  /* Precede lines by line number */
static uchar_t  rflag;                  /* Search directories recursively */
static uchar_t  bflag;                  /* Preccede matches by block number */
static uchar_t  sflag;                  /* Suppress file error messages */
static uchar_t  qflag;                  /* Suppress standard output */
static uchar_t  wflag;                  /* Search for expression as a word */
static uchar_t  xflag;                  /* Anchoring */
static uchar_t  Eflag;                  /* Egrep or -E flag */
static uchar_t  Fflag;                  /* Fgrep or -F flag */
static uchar_t  Rflag;                  /* Like rflag, but follow symlinks */
static uchar_t  outfn;                  /* Put out file name */
static char     *cmdname;

static int      use_wchar, use_bmg, mblocale;

static size_t   outbuflen, prntbuflen;
static char     *prntbuf;
static wchar_t  *outline;

static void     addfile(const char *fn);
static void     addpattern(char *s);
static void     fixpatterns(void);
static void     usage(void);
static int      grep(int, const char *);
static void     bmgcomp(char *, int);
static char     *bmgexec(char *, char *);
static int      recursive(const char *, const struct stat *, int, struct FTW *);
static void     process_path(const char *);
static void     process_file(const char *, int);

/*
 * mainline for grep
 */
```

```
128 int
129 main(int argc, char **argv)
130 {
131         char    *ap;
132         int     c;
133         int     fflag = 0;
134         int     i, n_pattern = 0, n_file = 0;
135         char    **pattern_list = NULL;
136         char    **file_list = NULL;

138         (void) setlocale(LC_ALL, "");
139 #if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
140 #define TEXT_DOMAIN     "SYS_TEST"      /* Use this only if it weren't */
141 #endif
142         (void) textdomain(TEXT_DOMAIN);

144         /*
145          * true if this is running on the multibyte locale
146          */
147         mblocale = (MB_CUR_MAX > 1);
148         /*
149          * Skip leading slashes
150          */
151         cmdname = argv[0];
152         if (ap = strrchr(cmdname, '/'))
153                 cmdname = ap + 1;

155         ap = cmdname;
156         /*
157          * Detect egrep/fgrep via command name, map to -E and -F options.
158          */
159         if (*ap == 'e' || *ap == 'E') {
160                 regflags |= REG_EXTENDED;
161                 egrep++;
162         } else {
163                 if (*ap == 'f' || *ap == 'F') {
164                         fgrep++;
165                 }
166         }

168         while ((c = getopt(argc, argv, "vwchHilnrbse:f:qxEFIR")) != EOF) {
 39         while ((c = getopt(argc, argv, "vwchilnrbse:f:qxEFIR")) != EOF) {
169                 switch (c) {
170                 case 'v':       /* POSIX: negate matches */
171                         nvflag = 0;
172                         break;

174                 case 'c':       /* POSIX: write count */
175                         cflag++;
176                         break;

178                 case 'i':       /* POSIX: ignore case */
179                         iflag++;
180                         regflags |= REG_ICASE;
181                         break;

183                 case 'l':       /* POSIX: Write filenames only */
184                         lflag++;
185                         break;

187                 case 'n':       /* POSIX: Write line numbers */
188                         nflag++;
189                         break;

191                 case 'r':       /* Solaris: search recursively */
192                         rflag++;
```



```
193                         break;

195                 case 'b':       /* Solaris: Write file block numbers */
196                         bflag++;
197                         break;

199                 case 's':       /* POSIX: No error msgs for files */
200                         sflag++;
201                         break;

203                 case 'e':       /* POSIX: pattern list */
204                         n_pattern++;
205                         pattern_list = realloc(pattern_list,
206                             sizeof (char *) * n_pattern);
207                         if (pattern_list == NULL) {
208                                 (void) fprintf(stderr,
209                                     gettext("%s: out of memory\n"),
210                                     cmdname);
211                                 exit(2);
212                         }
213                         *(pattern_list + n_pattern - 1) = optarg;
214                         break;

216                 case 'f':       /* POSIX: pattern file */
217                         fflag = 1;
218                         n_file++;
219                         file_list = realloc(file_list,
220                             sizeof (char *) * n_file);
221                         if (file_list == NULL) {
222                                 (void) fprintf(stderr,
223                                     gettext("%s: out of memory\n"),
224                                     cmdname);
225                                 exit(2);
226                         }
227                         *(file_list + n_file - 1) = optarg;
228                         break;

230                 /* based on options order h or H is set as in GNU grep */
231 #endif /* ! codereview */
232                 case 'h':       /* Solaris: supress printing of file name */
233                         hflag = 1;
234                         Hflag = 0;
235                         break;
236                 /* Solaris: precede every matching with file name */
237                 case 'H':
238                         Hflag = 1;
239                         hflag = 0;
240 #endif /* ! codereview */
241                         break;

243                 case 'q':       /* POSIX: quiet: status only */
244                         qflag++;
245                         break;

247                 case 'w':       /* Solaris: treat pattern as word */
248                         wflag++;
249                         break;

251                 case 'x':       /* POSIX: full line matches */
252                         xflag++;
253                         regflags |= REG_ANCHOR;
254                         break;

256                 case 'E':       /* POSIX: Extended RE's */
257                         regflags |= REG_EXTENDED;
258                         Eflag++;
```

```
259                         break;

261                 case 'F':       /* POSIX: strings, not RE's */
262                         Fflag++;
263                         break;

265                 case 'R':       /* Solaris: like rflag, but follow symlinks */
266                         Rflag++;
267                         rflag++;
268                         break;

270                 default:
271                         usage();
272                 }
273         }
274         /*
275          * If we're invoked as egrep or fgrep we need to do some checks
276          */

278         if (egrep || fgrep) {
279                 /*
280                  * Use of -E or -F with egrep or fgrep is illegal
281                  */
282                 if (Eflag || Fflag)
283                         usage();
284                 /*
285                  * Don't allow use of wflag with egrep / fgrep
286                  */
287                 if (wflag)
288                         usage();
289                 /*
290                  * For Solaris the -s flag is equivalent to XCU -q
291                  */
292                 if (sflag)
293                         qflag++;
294                 /*
295                  * done with above checks - set the appropriate flags
296                  */
297                 if (egrep)
298                         Eflag++;
299                 else                    /* Else fgrep */
300                         Fflag++;
301         }

303         if (wflag && (Eflag || Fflag)) {
304                 /*
305                  * -w cannot be specified with grep -F
306                  */
307                 usage();
308         }

310         /*
311          * -E and -F flags are mutually exclusive - check for this
312          */
313         if (Eflag && Fflag)
314                 usage();

316         /*
317          * -l overrides -H like in GNU grep
318          */
319         if (lflag)
320                 Hflag = 0;

322         /*
323 #endif /* ! codereview */
324          * -c, -l and -q flags are mutually exclusive
```



```
325          * We have -c override -l like in Solaris.
326          * -q overrides -l & -c programmatically in grep() function.
327          */
328         if (cflag && lflag)
329                 lflag = 0;

331         argv += optind - 1;
332         argc -= optind - 1;

334         /*
335          * Now handling -e and -f option
336          */
337         if (pattern_list) {
338                 for (i = 0; i < n_pattern; i++) {
339                         addpattern(pattern_list[i]);
340                 }
341                 free(pattern_list);
342         }
343         if (file_list) {
344                 for (i = 0; i < n_file; i++) {
345                         addfile(file_list[i]);
346                 }
347                 free(file_list);
348         }

350         /*
351          * No -e or -f?  Make sure there is one more arg, use it as the pattern.
352          */
353         if (patterns == NULL && !fflag) {
354                 if (argc < 2)
355                         usage();
356                 addpattern(argv[1]);
357                 argc--;
358                 argv++;
359         }

361         /*
362          * If -x flag is not specified or -i flag is specified
363          * with fgrep in a multibyte locale, need to use
364          * the wide character APIs.  Otherwise, byte-oriented
365          * process will be done.
366          */
367         use_wchar = Fflag && mblocale && (!xflag || iflag);

369         /*
370          * Compile Patterns and also decide if BMG can be used
371          */
372         fixpatterns();

374         /* Process all files: stdin, or rest of arg list */
375         if (argc < 2) {
376                 matched = grep(0, STDIN_FILENAME);
100                 matched = grep(0, gettext("(standard input)"));
377         } else {
378                 if (Hflag || (argc > 2 && hflag == 0))
102                 if (argc > 2 && hflag == 0)
379                         outfn = 1;      /* Print filename on match line */
380                 for (argv++; *argv != NULL; argv++) {
381                         process_path(*argv);
382                 }
383         }
384         /*
385          * Return() here is used instead of exit
386          */

388         (void) fflush(stdout);
```

```
390         if (errors)
391                 return (2);
392         return (matched ? 0 : 1);
393 }
```

*unchanged_portion_omitted_*

```
794 /*
795  * Do grep on a single file.
796  * Return true in any lines matched.
797  *
798  * We have two strategies:
799  * The fast one is used when we have a single pattern with
800  * a string known to occur in the pattern. We can then
801  * do a BMG match on the whole buffer.
802  * This is an order of magnitude faster.
803  * Otherwise we split the buffer into lines,
804  * and check for a match on each line.
805  */
806 static int
807 grep(int fd, const char *fn)
808 {
809         PATTERN *pp;
810         off_t   data_len;       /* length of the data chunk */
811         off_t   line_len;       /* length of the current line */
812         off_t   line_offset;    /* current line's offset from the beginning */
813         long long       lineno;
814         long long       matches = 0;    /* Number of matching lines */
815         int     newlinep;       /* 0 if the last line of file has no newline */
816         char    *ptr, *ptrend;


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

822         pp = patterns;

824         if (use_bmg) {
825                 bmgcomp(pp->pattern, strlen(pp->pattern));
826         }

828         if (use_wchar && outline == NULL) {
829                 outbuflen = BUFSIZE + 1;
830                 outline = malloc(sizeof (wchar_t) * outbuflen);
831                 if (outline == NULL) {
832                         (void) fprintf(stderr, gettext("%s: out of memory\n"),
833                             cmdname);
834                         exit(2);
835                 }
836         }

838         if (prntbuf == NULL) {
839                 prntbuflen = BUFSIZE;
840                 if ((prntbuf = malloc(prntbuflen + 1)) == NULL) {
841                         (void) fprintf(stderr, gettext("%s: out of memory\n"),
842                             cmdname);
843                         exit(2);
844                 }
845         }

847         line_offset = 0;
848         lineno = 0;
849         newlinep = 1;
850         data_len = 0;
851         for (; ; ) {
852                 long    count;
```



```
853                 off_t   offset = 0;

855                 if (data_len == 0) {
856                         /*
857                          * If no data in the buffer, reset ptr
858                          */
859                         ptr = prntbuf;
860                 }
861                 if (ptr == prntbuf) {
862                         /*
863                          * The current data chunk starts from prntbuf.
864                          * This means either the buffer has no data
865                          * or the buffer has no newline.
866                          * So, read more data from input.
867                          */
868                         count = read(fd, ptr + data_len, prntbuflen - data_len);
869                         if (count < 0) {
870                                 /* read error */
871                                 if (cflag) {
872                                         if (outfn && !rflag) {
873                                                 (void) fprintf(stdout,
874                                                     "%s:", fn);
875                                         }
876                                         if (!qflag && !rflag) {
877                                                 (void) fprintf(stdout, "%lld\n",
878                                                     matches);
879                                         }
880                                 }
881                                 return (0);
882                         } else if (count == 0) {
883                                 /* no new data */
884                                 if (data_len == 0) {
885                                         /* end of file already reached */
886                                         break;
887                                 }
888                                 /* last line of file has no newline */
889                                 ptrend = ptr + data_len;
890                                 newlinep = 0;
891                                 goto L_start_process;
892                         }
893                         offset = data_len;
894                         data_len += count;
895                 }

897                 /*
898                  * Look for newline in the chunk
899                  * between ptr + offset and ptr + data_len - offset.
900                  */
901                 ptrend = find_nl(ptr + offset, data_len - offset);
902                 if (ptrend == NULL) {
903                         /* no newline found in this chunk */
904                         if (ptr > prntbuf) {
905                                 /*
906                                  * Move remaining data to the beginning
907                                  * of the buffer.
908                                  * Remaining data lie from ptr for
909                                  * data_len bytes.
910                                  */
911                                 (void) memmove(prntbuf, ptr, data_len);
912                         }
913                         if (data_len == prntbuflen) {
914                                 /*
915                                  * No enough room in the buffer
916                                  */
917                                 prntbuflen += BUFSIZE;
918                                 prntbuf = realloc(prntbuf, prntbuflen + 1);
```

```
                                if (prntbuf == NULL) {
                                        (void) fprintf(stderr,
                                            gettext("%s: out of memory\n"),
                                            cmdname);
                                        exit(2);
                                }
                        }
                        ptr = prntbuf;
                        /* read the next input */
                        continue;
                }
L_start_process:

                /*
                 * Beginning of the chunk:      ptr
                 * End of the chunk:            ptr + data_len
                 * Beginning of the line:       ptr
                 * End of the line:             ptrend
                 */

                if (use_bmg) {
                        /*
                         * Use Boyer-Moore-Gosper algorithm to find out if
                         * this chunk (not this line) contains the specified
                         * pattern.  If not, restart from the last line
                         * of this chunk.
                         */
                        char    *bline;
                        bline = bmgexec(ptr, ptr + data_len);
                        if (bline == NULL) {
                                /*
                                 * No pattern found in this chunk.
                                 * Need to find the last line
                                 * in this chunk.
                                 */
                                ptrend = rfind_nl(ptr, data_len);

                                /*
                                 * When this chunk does not contain newline,
                                 * ptrend becomes NULL, which should happen
                                 * when the last line of file does not end
                                 * with a newline.  At such a point,
                                 * newlinep should have been set to 0.
                                 * Therefore, just after jumping to
                                 * L_skip_line, the main for-loop quits,
                                 * and the line_len value won't be
                                 * used.
                                 */
                                line_len = ptrend - ptr;
                                goto L_skip_line;
                        }
                        if (bline > ptrend) {
                                /*
                                 * Pattern found not in the first line
                                 * of this chunk.
                                 * Discard the first line.
                                 */
                                line_len = ptrend - ptr;
                                goto L_skip_line;
                        }
                        /*
                         * Pattern found in the first line of this chunk.
                         * Using this result.
                         */
                        *ptrend = '\0';
                        line_len = ptrend - ptr;
```



```
                        /*
                         * before jumping to L_next_line,
                         * need to handle xflag if specified
                         */
                        if (xflag && (line_len != bmglen ||
                            strcmp(bmgpat, ptr) != 0)) {
                                /* didn't match */
                                pp = NULL;
                        } else {
                                pp = patterns; /* to make it happen */
                        }
                        goto L_next_line;
                }
                lineno++;
                /*
                 * Line starts from ptr and ends at ptrend.
                 * line_len will be the length of the line.
                 */
                *ptrend = '\0';
                line_len = ptrend - ptr;

                /*
                 * From now, the process will be performed based
                 * on the line from ptr to ptrend.
                 */
                if (use_wchar) {
                        size_t  len;

                        if (line_len >= outbuflen) {
                                outbuflen = line_len + 1;
                                outline = realloc(outline,
                                    sizeof (wchar_t) * outbuflen);
                                if (outline == NULL) {
                                        (void) fprintf(stderr,
                                            gettext("%s: out of memory\n"),
                                            cmdname);
                                        exit(2);
                                }
                        }

                        len = mbstowcs(outline, ptr, line_len);
                        if (len == (size_t)-1) {
                                (void) fprintf(stderr, gettext(
        "%s: input file \"%s\": line %lld: invalid multibyte character\n"),
                                    cmdname, fn, lineno);
                                /* never match a line with invalid sequence */
                                goto L_skip_line;
                        }
                        outline[len] = L'\0';

                        if (iflag) {
                                wchar_t *cp;
                                for (cp = outline; *cp != '\0'; cp++) {
                                        *cp = towlower((wint_t)*cp);
                                }
                        }

                        if (xflag) {
                                for (pp = patterns; pp; pp = pp->next) {
                                        if (outline[0] == pp->wpattern[0] &&
                                            wcscmp(outline,
                                            pp->wpattern) == 0) {
                                                /* matched */
                                                break;
                                        }
```

```
1051                            }
1052                    } else {
1053                            for (pp = patterns; pp; pp = pp->next) {
1054                                    if (wcswcs(outline, pp->wpattern)
1055                                        != NULL) {
1056                                            /* matched */
1057                                            break;
1058                                    }
1059                            }
1060                    }
1061            } else if (Fflag) {
1062                    /* fgrep in byte-oriented handling */
1063                    char    *fptr;
1064                    if (iflag) {
1065                            fptr = istrdup(ptr);
1066                    } else {
1067                            fptr = ptr;
1068                    }
1069                    if (xflag) {
1070                            /* fgrep -x */
1071                            for (pp = patterns; pp; pp = pp->next) {
1072                                    if (fptr[0] == pp->pattern[0] &&
1073                                        strcmp(fptr, pp->pattern) == 0) {
1074                                            /* matched */
1075                                            break;
1076                                    }
1077                            }
1078                    } else {
1079                            for (pp = patterns; pp; pp = pp->next) {
1080                                    if (strstr(fptr, pp->pattern) != NULL) {
1081                                            /* matched */
1082                                            break;
1083                                    }
1084                            }
1085                    }
1086            } else {
1087                    /* grep or egrep */
1088                    for (pp = patterns; pp; pp = pp->next) {
1089                            int     rv;

1091                            rv = regexec(&pp->re, ptr, 0, NULL, 0);
1092                            if (rv == REG_OK) {
1093                                    /* matched */
1094                                    break;
1095                            }

1097                            switch (rv) {
1098                            case REG_NOMATCH:
1099                                    break;
1100                            case REG_ECHAR:
1101                                    (void) fprintf(stderr, gettext(
1102            "%s: input file \"%s\": line %lld: invalid multibyte character\n"),
1103                                        cmdname, fn, lineno);
1104                                    break;
1105                            default:
1106                                    (void) regerror(rv, &pp->re, errstr,
1107                                        sizeof (errstr));
1108                                    (void) fprintf(stderr, gettext(
1109            "%s: input file \"%s\": line %lld: %s\n"),
1110                                        cmdname, fn, lineno, errstr);
1111                                    exit(2);
1112                            }
1113                    }
1114            }

1116 L_next_line:
```



```
1117                /*
1118                 * Here, if pp points to non-NULL, something has been matched
1119                 * to the pattern.
1120                 */
1121                if (nvflag == (pp != NULL)) {
1122                        matches++;
1123                        /*
1124                         * Handle q, l, and c flags.
1125                         */
1126                        if (qflag) {
1127                                /* no need to continue */
1128                                /*
1129                                 * End of this line is ptrend.
1130                                 * We have read up to ptr + data_len.
1131                                 */
1132                                off_t   pos;
1133                                pos = ptr + data_len - (ptrend + 1);
1134                                (void) lseek(fd, -pos, SEEK_CUR);
1135                                exit(0);
1136                        }
1137                        if (lflag) {
1138                                (void) printf("%s\n", fn);
1139                                break;
1140                        }
1141                        if (!cflag) {
1142                                if (Hflag || outfn) {
 866                                if (outfn) {
1143                                        (void) printf("%s:", fn);
1144                                }
1145                                if (bflag) {
1146                                        (void) printf("%lld:", (offset_t)
1147                                            (line_offset / BSIZE));
1148                                }
1149                                if (nflag) {
1150                                        (void) printf("%lld:", lineno);
1151                                }
1152                                *ptrend = '\n';
1153                                (void) fwrite(ptr, 1, line_len + 1, stdout);
1154                        }
1155                        if (ferror(stdout)) {
1156                                return (0);
1157                        }
1158                }
1159 L_skip_line:
1160                if (!newlinep)
1161                        break;

1163                data_len -= line_len + 1;
1164                line_offset += line_len + 1;
1165                ptr = ptrend + 1;
1166        }

1168        if (cflag) {
1169                if (Hflag || outfn) {
 893                if (outfn) {
1170                        (void) printf("%s:", fn);
1171                }
1172                if (!qflag) {
1173                        (void) printf("%lld\n", matches);
1174                }
1175        }
1176        return (matches != 0);
1177 }

1179 /*
1180  * usage message for grep
```

```
 */
static void
usage(void)
{
        if (egrep || fgrep) {
                (void) fprintf(stderr, gettext("Usage:\t%s"), cmdname);
                (void) fprintf(stderr,
                    gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvx] "
                    gettext(" [-c|-l|-q] [-r|-R] [-bhinsvx] "
                    "pattern_list [file ...]\n"));

                (void) fprintf(stderr, "\t%s", cmdname);
                (void) fprintf(stderr,
                    gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvx] "
                    gettext(" [-c|-l|-q] [-r|-R] [-bhinsvx] "
                    "[-e pattern_list]... "
                    "[-f pattern_file]... [file...]\n"));
        } else {
                (void) fprintf(stderr, gettext("Usage:\t%s"), cmdname);
                (void) fprintf(stderr,
                    gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvwx] "
                    gettext(" [-c|-l|-q] [-r|-R] [-bhinsvwx] "
                    "pattern_list [file ...]\n"));

                (void) fprintf(stderr, "\t%s", cmdname);
                (void) fprintf(stderr,
                    gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvwx] "
                    gettext(" [-c|-l|-q] [-r|-R] [-bhinsvwx] "
                    "[-e pattern_list]... "
                    "[-f pattern_file]... [file...]\n"));

                (void) fprintf(stderr, "\t%s", cmdname);
                (void) fprintf(stderr,
                    gettext(" -E [-c|-l|-q] [-r|-R] [-bhHinsvx] "
                    gettext(" -E [-c|-l|-q] [-r|-R] [-bhinsvx] "
                    "pattern_list [file ...]\n"));

                (void) fprintf(stderr, "\t%s", cmdname);
                (void) fprintf(stderr,
                    gettext(" -E [-c|-l|-q] [-r|-R] [-bhHinsvx] "
                    gettext(" -E [-c|-l|-q] [-r|-R] [-bhinsvx] "
                    "[-e pattern_list]... "
                    "[-f pattern_file]... [file...]\n"));

                (void) fprintf(stderr, "\t%s", cmdname);
                (void) fprintf(stderr,
                    gettext(" -F [-c|-l|-q] [-r|-R] [-bhHinsvx] "
                    gettext(" -F [-c|-l|-q] [-r|-R] [-bhinsvx] "
                    "pattern_list [file ...]\n"));

                (void) fprintf(stderr, "\t%s", cmdname);
                (void) fprintf(stderr,
                    gettext(" -F [-c|-l|-q] [-bhHinsvx] [-e pattern_list]... "
                    gettext(" -F [-c|-l|-q] [-bhinsvx] [-e pattern_list]... "
                    "[-f pattern_file]... [file...]\n"));
        }
        exit(2);
        /* NOTREACHED */
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    9118 Thu May 30 19:29:53 2013
new/usr/src/man/man1/egrep.1
3737 grep does not support -H option
**********************************************************
'\" te
.\" Copyright 1989 AT&T
.\" Copyright (c) 2006, Sun Microsystems, Inc. All Rights Reserved
.\" Portions Copyright (c) 1992, X/Open Company Limited All Rights Reserved
.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
.\" http://www.opengroup.org/bookstore/.
.\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
.\"  This notice shall appear on any product containing this material.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH EGREP 1 "May 3, 2013"
.TH EGREP 1 "Mar 24, 2006"
.SH NAME
egrep \- search a file for a pattern using full regular expressions
.SH SYNOPSIS
.LP
.nf
\fB/usr/bin/egrep\fR [\fB-bcHhilnqsv\fR] \fB-e\fR \fIpattern_list\fR [\fIfile...
\fB/usr/bin/egrep\fR [\fB-bchilnsv\fR] \fB-e\fR \fIpattern_list\fR [\fIfile...\f
.fi

.LP
.nf
\fB/usr/bin/egrep\fR [\fB-bcHhilnqsv\fR] \fB-f\fR \fIfile\fR [\fIfile...\fR]
\fB/usr/bin/egrep\fR [\fB-bchilnsv\fR] \fB-f\fR \fIfile\fR [\fIfile...\fR]
.fi

.LP
.nf
\fB/usr/bin/egrep\fR [\fB-bcHhilnqsv\fR] \fIpattern\fR [\fIfile...\fR]
\fB/usr/bin/egrep\fR [\fB-bchilnsv\fR] \fIpattern\fR [\fIfile...\fR]
.fi

.LP
.nf
\fB/usr/xpg4/bin/egrep\fR [\fB-bcHhilnqsvx\fR] \fB-e\fR \fIpattern_list\fR [\fB-
\fB/usr/xpg4/bin/egrep\fR [\fB-bchilnqsvx\fR] \fB-e\fR \fIpattern_list\fR [\fB-f
     [\fIfile...\fR]
.fi

.LP
.nf
\fB/usr/xpg4/bin/egrep\fR [\fB-bcHhilnqsvx\fR] [\fB-e\fR \fIpattern_list\fR] \fB
\fB/usr/xpg4/bin/egrep\fR [\fB-bchilnqsvx\fR] [\fB-e\fR \fIpattern_list\fR] \fB-
     [\fIfile...\fR]
.fi

.LP
.nf
\fB/usr/xpg4/bin/egrep\fR [\fB-bcHhilnqsvx\fR] \fIpattern\fR [\fIfile...\fR]
\fB/usr/xpg4/bin/egrep\fR [\fB-bchilnqsvx\fR] \fIpattern\fR [\fIfile...\fR]
.fi

.SH DESCRIPTION
.sp
.LP
The \fBegrep\fR (\fIexpression grep\fR) utility searches files for a pattern of
characters and prints all lines that contain that pattern. \fBegrep\fR uses
full regular expressions (expressions that have string values that use the full
set of alphanumeric and special characters) to match the patterns. It uses a
```



```
fast deterministic algorithm that sometimes needs exponential space.
.sp
.LP
If no files are specified, \fBegrep\fR assumes standard input. Normally, each
line found is copied to the standard output. The file name is printed before
each line found if there is more than one input file.
.SS "/usr/bin/egrep"
.sp
.LP
The \fB/usr/bin/egrep\fR utility accepts full regular expressions as described
on the \fBregexp\fR(5) manual page, except for \fB\e(\fR and \fB\e)\fR,
\fB\e(\fR and \fB\e)\fR, \fB\e{\fR and \fB\e}\fR, \fB\e<\fR and \fB\e>\fR, and
\fB\en\fR, and with the addition of:
.RS +4
.TP
1.
A full regular expression followed by \fB+\fR that matches one or more
occurrences of the full regular expression.
.RE
.RS +4
.TP
2.
A full regular expression followed by \fB?\fR that matches 0 or 1
occurrences of the full regular expression.
.RE
.RS +4
.TP
3.
Full regular expressions separated by | or by a \fBNEWLINE\fR that match
strings that are matched by any of the expressions.
.RE
.RS +4
.TP
4.
A full regular expression that can be enclosed in parentheses \fB()\fRfor
grouping.
.RE
.sp
.LP
Be careful using the characters \fB$\fR, \fB*\fR, \fB[\fR, \fB^\fR, |, \fB(\fR,
\fB)\fR, and \fB\e\fR in \fIfull regular expression\fR, because they are also
meaningful to the shell. It is safest to enclose the entire \fIfull regular
expression\fR in single quotes (\fBa\'\fR\fBa\'\fR).
.sp
.LP
The order of precedence of operators is \fB[\|]\fR, then \fB*\|?\|+\fR, then
concatenation, then | and NEWLINE.
.SS "/usr/xpg4/bin/egrep"
.sp
.LP
The \fB/usr/xpg4/bin/egrep\fR utility uses the regular expressions described in
the \fBEXTENDED REGULAR EXPRESSIONS\fR section of the \fBregex\fR(5) manual
page.
.SH OPTIONS
.sp
.LP
The following options are supported for both \fB/usr/bin/egrep\fR and
\fB/usr/xpg4/bin/egrep\fR:
.sp
.ne 2
.na
\fB\fB-b\fR\fR
.ad
.RS 19n
Precede each line by the block number on which it was found. This can be useful
in locating block numbers by context (first block is 0).
```

```
.RE

.sp
.ne 2
.na
\fB\fB-c\fR\fR
.ad
.RS 19n
Print only a count of the lines that contain the pattern.
.RE

.sp
.ne 2
.na
\fB\fB-e\fR \fIpattern_list\fR\fR
.ad
.RS 19n
Search for a \fIpattern_list\fR (\fIfull regular expression\fR that begins with
a \fB\(mi\fR).
.RE

.sp
.ne 2
.na
\fB\fB-f\fR \fIfile\fR\fR
.ad
.RS 19n
Take the list of \fIfull\fR \fIregular\fR \fIexpressions\fR from \fIfile\fR.
.RE

.sp
.ne 2
.na
\fB\fB-H\fR\fR
.ad
.RS 19n
Precedes each line by the name of the file containing the matching line.
.RE

.sp
.ne 2
.na
#endif /* ! codereview */
\fB\fB-h\fR\fR
.ad
.RS 19n
Suppress printing of filenames when searching multiple files.
.RE

.sp
.ne 2
.na
\fB\fB-i\fR\fR
.ad
.RS 19n
Ignore upper/lower case distinction during comparisons.
.RE

.sp
.ne 2
.na
\fB\fB-l\fR\fR
.ad
.RS 19n
Print the names of files with matching lines once, separated by NEWLINEs. Does
not repeat the names of files when the pattern is found more than once.
```



```
.RE

.sp
.ne 2
.na
\fB\fB-n\fR\fR
.ad
.RS 19n
Precede each line by its line number in the file (first line is 1).
.RE

.sp
.ne 2
.na
\fB\fB-q\fR\fR
.ad
.RS 19n
Quiet. Does not write anything to the standard output, regardless of matching
lines. Exits with zero status if an input line is selected.
.RE

.sp
.ne 2
.na
#endif /* ! codereview */
\fB\fB-s\fR\fR
.ad
.RS 19n
Legacy equivalent of \fB-q\fR.
Work silently, that is, display nothing except error messages. This is useful
for checking the error status.
.RE

.sp
.ne 2
.na
\fB\fB-v\fR\fR
.ad
.RS 19n
Print all lines except those that contain the pattern.
.RE

.SS "/usr/xpg4/bin/egrep"
.sp
.LP
The following options are supported for \fB/usr/xpg4/bin/egrep\fR only:
.sp
.ne 2
.na
\fB\fB-q\fR\fR
.ad
.RS 6n
Quiet. Does not write anything to the standard output, regardless of matching
lines. Exits with zero status if an input line is selected.
.RE

.sp
.ne 2
.na
\fB\fB-x\fR\fR
.ad
.RS 6n
Consider only input lines that use all characters in the line to match an
entire fixed string or regular expression to be matching lines.
.RE
```

```
.SH OPERANDS
.sp
.LP
The following operands are supported:
.sp
.ne 2
.na
\fB\fIfile\fR\fR
.ad
.RS 8n
A path name of a file to be searched for the patterns. If no \fIfile\fR
operands are specified, the standard input is used.
.RE

.SS "/usr/bin/egrep"
.sp
.ne 2
.na
\fB\fIpattern\fR\fR
.ad
.RS 11n
Specify a pattern to be used during the search for input.
.RE

.SS "/usr/xpg4/bin/egrep"
.sp
.ne 2
.na
\fB\fIpattern\fR\fR
.ad
.RS 11n
Specify one or more patterns to be used during the search for input. This
operand is treated as if it were specified as \fB-e\fR\fIpattern_list.\fR.
.RE

.SH USAGE
.sp
.LP
See \fBlargefile\fR(5) for the description of the behavior of \fBegrep\fR when
encountering files greater than or equal to 2 Gbyte ( 2^31 bytes).
.SH ENVIRONMENT VARIABLES
.sp
.LP
See \fBenviron\fR(5) for descriptions of the following environment variables
that affect the execution of \fBegrep\fR: \fBLC_COLLATE\fR, \fBLC_CTYPE\fR,
\fBLC_MESSAGES\fR, and \fBNLSPATH\fR.
.SH EXIT STATUS
.sp
.LP
The following exit values are returned:
.sp
.ne 2
.na
\fB\fB0\fR\fR
.ad
.RS 5n
If any matches are found.
.RE

.sp
.ne 2
.na
\fB\fB1\fR\fR
.ad
.RS 5n
If no matches are found.
```



```
.RE

.sp
.ne 2
.na
\fB\fB2\fR\fR
.ad
.RS 5n
For syntax errors or inaccessible files (even if matches were found).
.RE

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
.SS "/usr/bin/egrep"
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
CSI     Not Enabled
.TE

.SS "/usr/xpg4/bin/egrep"
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
CSI     Enabled
.TE

.SH SEE ALSO
.sp
.LP
\fBfgrep\fR(1), \fBgrep\fR(1), \fBsed\fR(1), \fBsh\fR(1), \fBattributes\fR(5),
\fBenviron\fR(5), \fBlargefile\fR(5), \fBregex\fR(5), \fBregexp\fR(5),
\fBXPG4\fR(5)
.SH NOTES
.sp
.LP
Ideally there should be only one \fBgrep\fR command, but there is not a single
algorithm that spans a wide enough range of space-time trade-offs.
.sp
.LP
Lines are limited only by the size of the available virtual memory.
.SS "/usr/xpg4/bin/egrep"
.sp
.LP
The \fB/usr/xpg4/bin/egrep\fR utility is identical to \fB/usr/xpg4/bin/grep\fR
\fB-E\fR. See \fBgrep\fR(1). Portable applications should use
\fB/usr/xpg4/bin/grep\fR \fB-E\fR.
```

```
**********************************************************
    7742 Thu May 30 19:29:53 2013
new/usr/src/man/man1/fgrep.1
3737 grep does not support -H option
**********************************************************
```

```
'\" te
.\" Copyright 1989 AT&T
.\" Copyright (c) 2006, Sun Microsystems, Inc.  All Rights Reserved
.\" Portions Copyright (c) 1992, X/Open Company Limited  All Rights Reserved
.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
.\" http://www.opengroup.org/bookstore/.
.\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
.\"  This notice shall appear on any product containing this material.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH FGREP 1 "May 3, 2013"
.TH FGREP 1 "Mar 24, 2006"
.SH NAME
fgrep \- search a file for a fixed-character string
.SH SYNOPSIS
.LP
.nf
\fB/usr/bin/fgrep\fR [\fB-bcHhilnqsvx\fR] \fB-e\fR \fIpattern_list\fR [\fIfile..
\fB/usr/bin/fgrep\fR [\fB-bchilnsvx\fR] \fB-e\fR \fIpattern_list\fR [\fIfile...\
.fi

.LP
.nf
\fB/usr/bin/fgrep\fR [\fB-bcHhilnqsvx\fR] \fB-f\fR \fIfile\fR [\fIfile...\fR]
\fB/usr/bin/fgrep\fR [\fB-bchilnsvx\fR] \fB-f\fR \fIfile\fR [\fIfile...\fR]
.fi

.LP
.nf
\fB/usr/bin/fgrep\fR [\fB-bcHhilnqsvx\fR] \fIpattern\fR [\fIfile...\fR]
\fB/usr/bin/fgrep\fR [\fB-bchilnsvx\fR] \fIpattern\fR [\fIfile...\fR]
.fi

.LP
.nf
\fB/usr/xpg4/bin/fgrep\fR [\fB-bcHhilnqsvx\fR] \fB-e\fR \fIpattern_list\fR [\fB-
\fB/usr/xpg4/bin/fgrep\fR [\fB-bchilnqsvx\fR] \fB-e\fR \fIpattern_list\fR [\fB-f
     [\fIfile...\fR]
.fi

.LP
.nf
\fB/usr/xpg4/bin/fgrep\fR [\fB-bcHhilnqsvx\fR] [\fB-e\fR \fIpattern_list\fR] \fB
\fB/usr/xpg4/bin/fgrep\fR [\fB-bchilnqsvx\fR] [\fB-e\fR \fIpattern_list\fR] \fB-
     [\fIfile...\fR]
.fi

.LP
.nf
\fB/usr/xpg4/bin/fgrep\fR [\fB-bcHhilnqsvx\fR] \fIpattern\fR [\fIfile...\fR]
\fB/usr/xpg4/bin/fgrep\fR [\fB-bchilnqsvx\fR] \fIpattern\fR [\fIfile...\fR]
.fi

.SH DESCRIPTION
.sp
.LP
The \fBfgrep\fR (fast \fBgrep\fR) utility searches files for a character string
and prints all lines that contain that string. \fBfgrep\fR is different from
\fBgrep\fR(1) and from \fBegrep\fR(1) because it searches for a string, instead
of searching for a pattern that matches an expression. \fBfgrep\fR uses a fast
```



```
and compact algorithm.
.sp
.LP
The characters \fB$\fR, \fB*\fR, \fB[\fR, \fB^\fR, |, \fB(\fR, \fB)\fR, and
\fB\e\fR are interpreted literally by \fBfgrep\fR, that is, \fBfgrep\fR does
not recognize full regular expressions as does \fBegrep\fR. These characters
have special meaning to the shell. Therefore, to be safe, enclose the entire
\fIstring\fR within single quotes (\fBa\'\fR).
.sp
.LP
If no files are specified, \fBfgrep\fR assumes standard input. Normally, each
line that is found is copied to the standard output. The file name is printed
before each line that is found if there is more than one input file.
.SH OPTIONS
.sp
.LP
The following options are supported for both \fB/usr/bin/fgrep\fR and
\fB/usr/xpg4/bin/fgrep\fR:
.sp
.ne 2
.na
\fB\fB-b\fR\fR
.ad
.RS 19n
Precedes each line by the block number on which the line was found. This can be
useful in locating block numbers by context. The first block is 0.
.RE

.sp
.ne 2
.na
\fB\fB-c\fR\fR
.ad
.RS 19n
Prints only a count of the lines that contain the pattern.
.RE

.sp
.ne 2
.na
\fB\fB-e\fR \fIpattern_list\fR\fR
.ad
.RS 19n
Searches for a \fIstring\fR in \fIpattern-list\fR. This is useful when the
\fIstring\fR begins with a \fB\(mi\fR\&.
.RE

.sp
.ne 2
.na
\fB\fB-f\fR \fIpattern-file\fR\fR
.ad
.RS 19n
Takes the list of patterns from \fIpattern-file\fR.
.RE

.sp
.ne 2
.na
\fB\fB-H\fR\fR
.ad
.RS 19n
Precedes each line by the name of the file containing the matching line.
.RE

.sp
```

```
.ne 2
.na
#endif /* ! codereview */
\fB\fB-h\fR\fR
.ad
.RS 19n
Suppresses printing of files when searching multiple files.
.RE

.sp
.ne 2
.na
\fB\fB-i\fR\fR
.ad
.RS 19n
Ignores upper/lower case distinction during comparisons.
.RE

.sp
.ne 2
.na
\fB\fB-l\fR\fR
.ad
.RS 19n
Prints the names of files with matching lines once, separated by new-lines.
Does not repeat the names of files when the pattern is found more than once.
.RE

.sp
.ne 2
.na
\fB\fB-n\fR\fR
.ad
.RS 19n
Precedes each line by its line number in the file. The first line is 1.
.RE

.sp
.ne 2
.na
\fB\fB-q\fR\fR
\fB\fB-s\fR\fR
.ad
.RS 19n
Quiet. Does not write anything to the standard output, regardless of matching
lines. Exits with zero status if an input line is selected.
Works silently, that is, displays nothing except error messages. This is useful
for checking the error status.
.RE

.sp
.ne 2
.na
\fB\fB-s\fR\fR
\fB\fB-v\fR\fR
.ad
.RS 19n
Legacy equivalent of \fB-q\fR.
Prints all lines except those that contain the pattern.
.RE

.sp
.ne 2
.na
\fB\fB-v\fR\fR
\fB\fB-x\fR\fR
```



```
.ad
.RS 19n
Prints all lines except those that contain the pattern.
Prints only lines that are matched entirely.
.RE

.SS "/usr/xpg4/bin/fgrep"
.sp
.LP
The following options are supported for \fB/usr/xpg4/bin/fgrep\fR only:
.sp
.ne 2
.na
\fB\fB-x\fR\fR
\fB\fB-q\fR\fR
.ad
.RS 19n
Prints only lines that are matched entirely.
.RS 6n
Quiet. Does not write anything to the standard output, regardless of matching
lines. Exits with zero status if an input line is selected.
.RE

.SH OPERANDS
.sp
.LP
The following operands are supported:
.sp
.ne 2
.na
\fB\fIfile\fR\fR
.ad
.RS 8n
Specifies a path name of a file to be searched for the patterns. If no
\fIfile\fR operands are specified, the standard input will be used.
.RE

.SS "/usr/bin/fgrep"
.sp
.ne 2
.na
\fB\fIpattern\fR\fR
.ad
.RS 11n
Specifies a pattern to be used during the search for input.
.RE

.SS "/usr/xpg4/bin/fgrep"
.sp
.ne 2
.na
\fB\fIpattern\fR\fR
.ad
.RS 11n
Specifies one or more patterns to be used during the search for input. This
operand is treated as if it were specified as \fB-e\fR \fIpattern_list\fR.
.RE

.SH USAGE
.sp
.LP
See \fBlargefile\fR(5) for the description of the behavior of \fBfgrep\fR when
encountering files greater than or equal to 2 Gbyte ( 2^31 bytes).
.SH ENVIRONMENT VARIABLES
.sp
.LP
```

```
238 See \fBenviron\fR(5) for descriptions of the following environment variables
239 that affect the execution of \fBfgrep\fR: \fBLC_COLLATE\fR, \fBLC_CTYPE\fR,
240 \fBLC_MESSAGES\fR, and \fBNLSPATH\fR.
241 .SH EXIT STATUS
242 .sp
243 .LP
244 The following exit values are returned:
245 .sp
246 .ne 2
247 .na
248 \fB\fB0\fR\fR
249 .ad
250 .RS 5n
251 If any matches are found
252 .RE

254 .sp
255 .ne 2
256 .na
257 \fB\fB1\fR\fR
258 .ad
259 .RS 5n
260 If no matches are found
261 .RE

263 .sp
264 .ne 2
265 .na
266 \fB\fB2\fR\fR
267 .ad
268 .RS 5n
269 For syntax errors or inaccessible files, even if matches were found.
270 .RE

272 .SS "/usr/xpg4/bin/fgrep"
273 .sp

275 .SH ATTRIBUTES
276 .sp
277 .LP
278 See \fBattributes\fR(5) for descriptions of the following attributes:
279 .sp
280 .TS
281 box;
282 c | c
283 l | l .
284 ATTRIBUTE TYPE  ATTRIBUTE VALUE
285 _
286 CSI     Enabled
287 .TE

289 .SH SEE ALSO
290 .sp
291 .LP
292 \fBed\fR(1), \fBegrep\fR(1), \fBgrep\fR(1), \fBsed\fR(1), \fBsh\fR(1),
293 \fBattributes\fR(5), \fBenviron\fR(5), \fBlargefile\fR(5), \fBXPG4\fR(5)
294 .SH NOTES
295 .sp
296 .LP
297 Ideally, there should be only one \fBgrep\fR command, but there is not a single
298 algorithm that spans a wide enough range of space-time tradeoffs.
299 .sp
300 .LP
301 Lines are limited only by the size of the available virtual memory.
302 .SS "/usr/xpg4/bin/fgrep"
303 .sp
```



```
304 .LP
305 The \fB/usr/xpg4/bin/fgrep\fR utility is identical to \fB/usr/xpg4/bin/grep\fR
306 \fB-F\fR (see \fBgrep\fR(1)). Portable applications should use
307 \fB/usr/xpg4/bin/grep\fR \fB-F\fR.
```

```
**********************************************************
   14031 Thu May 30 19:29:53 2013
new/usr/src/man/man1/grep.1
3737 grep does not support -H option
**********************************************************
'\" te
.\" Copyright 2012 Nexenta Systems, Inc. All rights reserved.
.\" Copyright 1989 AT&T
.\" Copyright (c) 2008, Sun Microsystems, Inc.  All Rights Reserved
.\" Portions Copyright (c) 1992, X/Open Company Limited  All Rights Reserved
.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
.\" http://www.opengroup.org/bookstore/.
.\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
.\"  This notice shall appear on any product containing this material.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH GREP 1 "May 3, 2013"
.TH GREP 1 "Feb 26, 2008"
.SH NAME
grep \- search a file for a pattern
.SH SYNOPSIS
.LP
.nf
\fB/usr/bin/grep\fR [\fB-c\fR | \fB-l\fR |\fB-q\fR] [\fB-r\fR | \fB-R\fR] [\fB-b
\fB/usr/bin/grep\fR [\fB-c\fR | \fB-l\fR |\fB-q\fR] [\fB-r\fR | \fB-R\fR] [\fB-b
     \fIlimited-regular-expression\fR [\fIfilename\fR]...
.fi

.LP
.nf
\fB/usr/xpg4/bin/grep\fR [\fB-E\fR | \fB-F\fR] [\fB-c\fR | \fB-l\fR | \fB-q\fR]
     [\fB-bHhinsvwx\fR] \fB-e\fR \fIpattern_list\fR... [\fB-f\fR \fIpattern_file\
     [\fB-bhinsvwx\fR] \fB-e\fR \fIpattern_list\fR... [\fB-f\fR \fIpattern_file\f
     [\fIfile\fR]...
.fi

.LP
.nf
\fB/usr/xpg4/bin/grep\fR [\fB-E\fR | \fB-F\fR] [\fB-c\fR | \fB-l\fR | \fB-q\fR]
     [\fB-bHhinsvwx\fR] [\fB-e\fR \fIpattern_list\fR]... \fB-f\fR \fIpattern_file
     [\fB-bhinsvwx\fR] [\fB-e\fR \fIpattern_list\fR]... \fB-f\fR \fIpattern_file\
     [\fIfile\fR]...
.fi

.LP
.nf
\fB/usr/xpg4/bin/grep\fR [\fB-E\fR | \fB-F\fR] [\fB-c\fR | \fB-l\fR | \fB-q\fR]
     [\fB-bHhinsvwx\fR] \fIpattern\fR [\fIfile\fR]...
     [\fB-bhinsvwx\fR] \fIpattern\fR [\fIfile\fR]...
.fi

.SH DESCRIPTION
.sp
.LP
The \fBgrep\fR utility searches text files for a pattern and prints all lines
that contain that pattern.  It uses a compact non-deterministic algorithm.
.sp
.LP
Be careful using the characters \fB$\fR, \fB*\fR, \fB[\fR, \fB^\fR, \fB|\fR,
\fB(\fR, \fB)\fR, and \fB\e\fR in the \fIpattern_list\fR because they are also
meaningful to the shell. It is safest to enclose the entire \fIpattern_list\fR
in single quotes \fBa'\fR\&...\fBa'\fR\&.
.sp
.LP
If no files are specified, \fBgrep\fR assumes standard input. Normally, each
```



```
line found is copied to standard output. The file name is printed before each
line found if there is more than one input file.
.SS "/usr/bin/grep"
.sp
.LP
The \fB/usr/bin/grep\fR utility uses limited regular expressions like those
described on the \fBregexp\fR(5) manual page to match the patterns.
.SS "/usr/xpg4/bin/grep"
.sp
.LP
The options \fB-E\fR and \fB-F\fR affect the way \fB/usr/xpg4/bin/grep\fR
interprets \fIpattern_list\fR. If \fB-E\fR is specified,
\fB/usr/xpg4/bin/grep\fR interprets \fIpattern_list\fR as a full regular
expression (see \fB-E\fR for description).  If \fB-F\fR is specified,
\fBgrep\fR interprets \fIpattern_list\fR as a fixed string. If neither are
specified, \fBgrep\fR interprets \fIpattern_list\fR as a basic regular
expression as described on \fBregex\fR(5) manual page.
.SH OPTIONS
.sp
.LP
The following options are supported for both \fB/usr/bin/grep\fR and
\fB/usr/xpg4/bin/grep\fR:
.sp
.ne 2
.na
\fB\fB-b\fR\fR
.ad
.RS 6n
Precedes each line by the block number on which it was found. This can be
useful in locating block numbers by context (first block is 0).
.RE

.sp
.ne 2
.na
\fB\fB-c\fR\fR
.ad
.RS 6n
Prints only a count of the lines that contain the pattern.
.RE

.sp
.ne 2
.na
\fB\fB-H\fR\fR
.ad
.RS 6n
Precedes each line by the name of the file containing the matching line.
.RE

.sp
.ne 2
.na
#endif /* ! codereview */
\fB\fB-h\fR\fR
.ad
.RS 6n
Prevents the name of the file containing the matching line from being prepended
to that line.  Used when searching multiple files.
.RE

.sp
.ne 2
.na
\fB\fB-i\fR\fR
.ad
```

```
.RS 6n
Ignores upper/lower case distinction during comparisons.
.RE

.sp
.ne 2
.na
\fB\fB-l\fR\fR
.ad
.RS 6n
Prints only the names of files with matching lines, separated by NEWLINE
characters.  Does not repeat the names of files when the pattern is found more
than once.
.RE

.sp
.ne 2
.na
\fB\fB-n\fR\fR
.ad
.RS 6n
Precedes each line by its line number in the file (first line is 1).
.RE

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

.sp
.ne 2
.na
\fB\fB-R\fR\fR
.ad
.RS 6n
Read all files under each directory, recursively, following all symbolic links.
.RE

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

.sp
.ne 2
.na
\fB\fB-s\fR\fR
.ad
.RS 6n
Suppresses error messages about nonexistent or unreadable files.
.RE

.sp
.ne 2
.na
```



```
\fB\fB-v\fR\fR
.ad
.RS 6n
Prints all lines except those that contain the pattern.
.RE

.sp
.ne 2
.na
\fB\fB-w\fR\fR
.ad
.RS 6n
Searches for the expression as a word as if surrounded by \fB\e<\fR and
\fB\e>\fR\&.
.RE

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

.sp
.ne 2
.na
\fB\fB-E\fR\fR
.ad
.RS 19n
Matches using full regular expressions. Treats each pattern specified as a full
regular expression. If any entire full regular expression pattern matches an
input line, the line is matched. A null full regular expression matches every
line. Each pattern is interpreted as a full regular expression as described on
the \fBregex\fR(5) manual page, except for \fB\e(\fR and \fB\e)\fR, and
including:
.RS +4
.TP
1.
A full regular expression followed by \fB+\fR that matches one or more
occurrences of the full regular expression.
.RE
.RS +4
.TP
2.
A full regular expression followed by \fB?\fR that matches 0 or 1
occurrences of the full regular expression.
.RE
.RS +4
.TP
3.
Full regular expressions separated by | or by a new-line that match strings
that are matched by any of the expressions.
.RE
.RS +4
```

```
.TP
4.
A full regular expression that is enclosed in parentheses \fB()\fR for
grouping.
.RE
The order of precedence of operators is \fB[\|]\fR, then \fB*\|?\|+\fR, then
concatenation, then | and new-line.
.RE

.sp
.ne 2
.na
\fB\fB-f\fR \fIpattern_file\fR\fR
.ad
.RS 19n
Reads one or more patterns from the file named by the path name
\fIpattern_file\fR. Patterns in \fIpattern_file\fR are terminated by a NEWLINE
character. A null pattern can be specified by an empty line in
\fIpattern_file\fR. Unless the \fB-E\fR or \fB-F\fR option is also specified,
each pattern is treated as a basic regular expression.
.RE

.sp
.ne 2
.na
\fB\fB-F\fR\fR
.ad
.RS 19n
Matches using fixed strings. Treats each pattern specified as a string instead
of a regular expression. If an input line contains any of the patterns as a
contiguous sequence of bytes, the line is matched. A null string matches every
line. See \fBfgrep\fR(1) for more information.
.RE

.sp
.ne 2
.na
\fB\fB-x\fR\fR
.ad
.RS 19n
Considers only input lines that use all characters in the line to match an
entire fixed string or regular expression to be matching lines.
.RE

.SH OPERANDS
.sp
.LP
The following operands are supported:
.sp
.ne 2
.na
\fB\fIfile\fR\fR
.ad
.RS 8n
A path name of a file to be searched for the patterns. If no \fIfile\fR
operands are specified, the standard input is used.
.RE

.SS "/usr/bin/grep"
.sp
.ne 2
.na
\fB\fIpattern\fR\fR
.ad
.RS 11n
Specifies a pattern to be used during the search for input.
```



```
.RE

.SS "/usr/xpg4/bin/grep"
.sp
.ne 2
.na
\fB\fIpattern\fR\fR
.ad
.RS 11n
Specifies one or more patterns to be used during the search for input. This
operand is treated as if it were specified as \fB-e\fR \fIpattern_list\fR.
.RE

.SH USAGE
.sp
.LP
The \fB-e\fR \fIpattern_list\fR option has the same effect as the
\fIpattern_list\fR operand, but is useful when \fIpattern_list\fR begins with
the hyphen delimiter. It is also useful when it is more convenient to provide
multiple patterns as separate arguments.
.sp
.LP
Multiple \fB-e\fR and \fB-f\fR options are accepted and \fBgrep\fR uses all of
the patterns it is given while matching input text lines. Notice that the order
of evaluation is not specified. If an implementation finds a null string as a
pattern, it is allowed to use that pattern first, matching every line, and
effectively ignore any other patterns.
.sp
.LP
The \fB-q\fR option provides a means of easily determining whether or not a
pattern (or string) exists in a group of files. When searching several files,
it provides a performance improvement (because it can quit as soon as it finds
the first match) and requires less care by the user in choosing the set of
files to supply as arguments (because it exits zero if it finds a match even if
\fBgrep\fR detected an access or read error on earlier file operands).
.SS "Large File Behavior"
.sp
.LP
See \fBlargefile\fR(5) for the description of the behavior of \fBgrep\fR when
encountering files greater than or equal to 2 Gbyte ( 2^31 bytes).
.SH EXAMPLES
.LP
\fBExample 1 \fRFinding All Uses of a Word
.sp
.LP
To find all uses of the word "\fBPosix\fR" (in any case) in the file
\fBtext.mm\fR, and write with line numbers:

.sp
.in +2
.nf
example% \fB/usr/bin/grep -i -n posix text.mm\fR
.fi
.in -2
.sp

.LP
\fBExample 2 \fRFinding All Empty Lines
.sp
.LP
To find all empty lines in the standard input:

.sp
.in +2
.nf
example% \fB/usr/bin/grep ^$\fR
```

```
.fi
.in -2
.sp

.sp
.LP
or

.sp
.in +2
.nf
example% \fB/usr/bin/grep -v .\fR
.fi
.in -2
.sp

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

.sp
.in +2
.nf
example% \fB/usr/xpg4/bin/grep 'abc
def'\fR
example% \fB/usr/xpg4/bin/grep -e 'abc
def'\fR
example% \fB/usr/xpg4/bin/grep -e 'abc' -e 'def'\fR
example% \fB/usr/xpg4/bin/grep -E 'abc|def'\fR
example% \fB/usr/xpg4/bin/grep -E -e 'abc|def'\fR
example% \fB/usr/xpg4/bin/grep -E -e 'abc' -e 'def'\fR
example% \fB/usr/xpg4/bin/grep -E 'abc
def'\fR
example% \fB/usr/xpg4/bin/grep -E -e 'abc
def'\fR
example% \fB/usr/xpg4/bin/grep -F -e 'abc' -e 'def'\fR
example% \fB/usr/xpg4/bin/grep -F 'abc
def'\fR
example% \fB/usr/xpg4/bin/grep -F -e 'abc
def'\fR
.fi
.in -2
.sp

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

.sp
.in +2
.nf
example% \fB/usr/xpg4/bin/grep -E '^abc$ ^def$'\fR
example% \fB/usr/xpg4/bin/grep -F -x 'abc def'\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 \fBgrep\fR: \fBLANG\fR, \fBLC_ALL\fR,
\fBLC_COLLATE\fR, \fBLC_CTYPE\fR, \fBLC_MESSAGES\fR, and \fBNLSPATH\fR.
.SH EXIT STATUS
.sp
.LP
The following exit values are returned:
.sp
.ne 2
.na
\fB\fB0\fR\fR
.ad
.RS 5n
One or more matches were found.
.RE

.sp
.ne 2
.na
\fB\fB1\fR\fR
.ad
.RS 5n
No matches were found.
.RE

.sp
.ne 2
.na
\fB\fB2\fR\fR
.ad
.RS 5n
Syntax errors or inaccessible files (even if matches were found).
.RE

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

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
CSI     Not Enabled
.TE

.SS "/usr/xpg4/bin/grep"
.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
\fBegrep\fR(1), \fBfgrep\fR(1), \fBsed\fR(1), \fBsh\fR(1), \fBattributes\fR(5),
\fBenviron\fR(5), \fBlargefile\fR(5), \fBregex\fR(5), \fBregexp\fR(5),
\fBstandards\fR(5)
.SH NOTES
.SS "/usr/bin/grep"
.sp
.LP
Lines are limited only by the size of the available virtual memory. If there is
a line with embedded nulls, \fBgrep\fR only matches up to the first null. If
the line matches, the entire line is printed.
.SS "/usr/xpg4/bin/grep"
.sp
.LP
The results are unspecified if input files contain lines longer than
\fBLINE_MAX\fR bytes or contain binary data. \fBLINE_MAX\fR is defined in
\fB/usr/include/limits.h\fR.
```