```
**********************************************************
   26088 Thu May 23 21:03:49 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   */

%{
#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;
int     nfile;
long long blkno;
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':
```

```
                        if (!lflag) /* H is excluded by l as in GNU grep */
                                Hflag++;
                        hflag = 0; /* H excludes h */
                        continue;

#endif /* ! codereview */
                case 'h':
                        hflag++;
                        Hflag = 0; /* h excludes H */
#endif /* ! codereview */
                        continue;

                case 'y':
                case 'i':
                        iflag++;
                        continue;

                case 'l':
                        lflag++;
                        Hflag = 0; /* l excludes H */
#endif /* ! codereview */
                        continue;

                case 'n':
                        nflag++;
                        continue;

                case 'q':
                case 's': /* Solaris: legacy option */
                        qflag++;
                case 's':
                        sflag++;
                        continue;

                case 'v':
                        vflag++;
                        continue;

                case '?':
                        errflag++;
                }
        if (errflag || ((argc <= 0) && !fflag && !eflag)) {
                fprintf(stderr, gettext("usage: egrep %s\n"), gettext(USAGE));
                exit(2);
        }
        if(!eflag && !fflag) {
                input = argv[optind];
                optind++;
        }

        argc -= optind;
        argv = &argv[optind];

        /* allocate initial space for arrays */
        if((name = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
                overflo();
        if((left = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
                overflo();
        if((right = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
                overflo();
        if((parent = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
                overflo();
        if((foll = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
                overflo();
        if((tmpstat = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
                overflo();
```



```
        if((initstat = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
                overflo();
        if((chars = (char *)malloc(MAXLIN)) == (char *)0)
                overflo();
        if((lower = (wchar_t *)malloc(MAXLIN*sizeof(wchar_t))) == (wchar_t *)0)
                overflo();
        if((upper = (wchar_t *)malloc(MAXLIN*sizeof(wchar_t))) == (wchar_t *)0)
                overflo();
        if((positions = (int *)malloc(MAXPOS*sizeof(int))) == (int *)0)
                overflo();
        maxlin = MAXLIN;
        maxclin = MAXLIN;
        maxwclin = MAXLIN;
        maxpos = MAXPOS;

        yyparse();

        cfoll(line-1);
        cgotofn();
        nfile = argc;
        if (argc<=0) {
                execute(0);
        }
        else while (--argc >= 0) {
                if (reinit == 1) clearg();
                execute(*argv++);
        }
        return (badbotch ? 2 : nsucc==0);
}

void
execute(char *file)
{
        char *p;
        int cstat;
        wchar_t c;
        int t;
        long count;
        long count1, count2;
        long nchars;
        int succ;
        char *ptr, *ptrend, *lastptr;
        char *buf;
        long lBufSiz;
        FILE *f;
        int nlflag;

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

        if (file) {
                if ((f = fopen(file, "r")) == NULL) {
                        fprintf(stderr,
                                gettext("egrep: can't open %s\n"), file);
                        badbotch=1;
                        return;
                }
        } else {
                file = "<stdin>";
                f = stdin;
                file = STDIN_FILENAME;
#endif /* ! codereview */
                }
                lnum = 1;
```

```
824         tln = 0;
825         if((count = read(fileno(f), buf, EBUFSIZ)) <= 0) {
826                 fclose(f);

828                 if (cflag && !qflag) {
829                         if (Hflag || (nfile > 1 && !hflag))
734                 if (cflag) {
735                         if (nfile>1 && !hflag)
830                                 fprintf(stdout, "%s:", file);
831                         fprintf(stdout, "%lld\n", tln);
832                 }
833                 return;
834         }

836         blkno = count;
837         ptr = buf;
838         for(;;) {
839                 if((ptrend = memchr(ptr, '\n', buf + count - ptr)) == NULL) {
840                         /*
841                                 move the unused partial record to the head of th
842                         */
843                         if (ptr > buf) {
844                                 count = buf + count - ptr;
845                                 memmove (buf, ptr, count);
846                                 ptr = buf;
847                         }

849                         /*
850                                 Get a bigger buffer if this one is full
851                         */
852                         if(count > lBufSiz) {
853                                 /*
854                                         expand the buffer
855                                 */
856                                 lBufSiz += EBUFSIZ;
857                                 if ((buf = realloc (buf, lBufSiz + EBUFSIZ)) ==
858                                         exit (2); /* out of memory - BAIL */
859                                 }

861                                 ptr = buf;
862                         }

864                         p = buf + count;
865                         if((count1 = read(fileno(f), p, EBUFSIZ)) > 0) {
866                                 count += count1;
867                                 blkno += count1;
868                                 continue;
869                         }
870                         ptrend = ptr + count;
871                         nlflag = 0;
872                 } else
873                         nlflag = 1;
874                 *ptrend = '\n';
875                 p = ptr;
876                 lastptr = ptr;
877                 cstat = istat;
878                 succ = 0;
879                 for(;;) {
880                         if(out[cstat]) {
881                                 if(multibyte && p > ptr) {
882                                         wchar_t wchar;
883                                         int length;
884                                         char *endptr = p;
885                                         p = lastptr;
886                                         while(p < endptr) {
887                                                 length = mbtowc(&wchar, p, MB_LE
```



```
888                                                 if(length <= 1)
889                                                         p++;
890                                                 else
891                                                         p += length;
892                                         }
893                                         if(p == endptr) {
894                                                 succ = !vflag;
895                                                 break;
896                                         }
897                                         cstat = 1;
898                                         length = mbtowc(&wchar, lastptr, MB_LEN_
899                                         if(length <= 1)
900                                                 lastptr++;
901                                         else
902                                                 lastptr += length;
903                                         p = lastptr;
904                                         continue;
905                                 }
906                                 succ = !vflag;
907                                 break;
908                         }
909                         c = (unsigned char)*p++;
910                         if ((t = gotofn[cstat][c]) == 0)
911                                 cstat = nxtst(cstat, c);
912                         else
913                                 cstat = t;
914                         if(c == RIGHT) {
915                                 if(out[cstat]) {
916                                         succ = !vflag;
917                                         break;
918                                 }
919                                 succ = vflag;
920                                 break;
921                         }
922                 }
923                 if (succ) {
829                 if(succ) {
924                         nsucc = 1;
925                         if (lflag || qflag) {
926                                 if (!qflag)
927                                         (void) printf("%s\n", file);
831                         if (cflag) tln++;
832                         else if (sflag)
833                                 ;       /* ugh */
834                         else if (lflag) {
835                                 printf("%s\n", file);
928                                 fclose(f);
929                                 return;
930                         }
931                         if (cflag) {
932                                 tln++;
933                         } else {
934                                 if (Hflag || (nfile > 1 && !hflag))
935                                         printf("%s:", file);
839                         else {
840                                 if (nfile > 1 && !hflag)
841                                         printf(gettext("%s:"), file);
936                                 if (bflag) {
937                                         nchars = blkno - (buf + count - ptrend)
938                                         if(nlflag)
939                                                 nchars++;
940                                         printf("%lld:", nchars/BLKSIZE);
941                                 }
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_*

```
**********************************************************
   14418 Thu May 23 21:03:50 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   */

#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;
int     Hflag, hflag, iflag;
int     bflag, cflag, lflag, fflag, nflag, vflag, xflag, eflag, sflag;
int     hflag, iflag;
```

```
126 int     retcode = 0;
127 int     nfile;
128 long long blkno;
129 int     nsucc;
130 long long tln;
131 FILE    *wordf;
132 char    *argptr;
133 off_t input_size = 0;

135 void    execute(char *);
136 void    cgotofn(void);
137 void    overflo(void);
138 void    cfail(void);

140 static long fw_lBufsiz = 0;

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

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

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

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

175                 case 'i':
176                 case 'y':
177                         iflag++;
178                         continue;

180                 case 'c':
181                         cflag++;
182                         continue;

184                 case 'e':
185                         eflag++;
186                         argptr = optarg;
187                         input_size = strlen(argptr);
188                         continue;
```



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

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

209                         continue;

211                 case 'l':
212                         lflag++;
213                         continue;

215                 case 'n':
216                         nflag++;
217                         continue;

219                 case 'v':
220                         vflag++;
221                         continue;

223                 case 'x':
224                         xflag++;
225                         continue;

227                 case '?':
228                         errflg++;
229         }

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

245 /*
246  * Normally we need one struct words for each letter in the pattern
247  * plus one terminating struct words with outp = 1, but when -x option
248  * is specified we require one more struct words for '\n' character so we
249  * calculate the input_size as below. We add extra 1 because
250  * (input_size/2) rounds off odd numbers
251  */

253         if (xflag) {
```

```
254                 input_size = input_size + (input_size/2) + 1;
255         }
256
257         input_size++;
258
259         w = (struct words *)calloc(input_size, sizeof (struct words));
260         if (w == NULL) {
261                 (void) fprintf(stderr,
262                         gettext("fgrep: could not allocate "
263                                 "memory for wordlist\n"));
264                 exit(2);
265         }
266
267         getwidth(&WW);
268         if ((WIDTH1 == 0) && (WIDTH2 == 0) &&
269                 (WIDTH3 == 0)) {
270                 /*
271                  * If non EUC-based locale,
272                  * assume WIDTH1 is 1.
273                  */
274                 WIDTH1 = 1;
275         }
276         WIDTH2++;
277         WIDTH3++;
278
279         cgotofn();
280         cfail();
281         nfile = argc;
282         argv = &argv[optind];
283         if (argc <= 0) {
284                 execute((char *)NULL);
285         } else
286                 while (--argc >= 0) {
287                         execute(*argv);
288                         argv++;
289                 }
290
291         if (w != NULL) {
292                 free(w);
293         }
294
295         return (retcode != 0 ? retcode : nsucc == 0);
296 }
297
298 void
299 execute(char *file)
300 {
301         char *p;
302         struct words *c;
303         int ccount;
304         static char *buf = NULL;
305         int failed;
306         char *nlp;
307         wchar_t lc;
308         int cw;
309
310         if (buf == NULL) {
311                 fw_lBufsiz = BUFSIZ;
312                 if ((buf = malloc(fw_lBufsiz + BUFSIZ)) == NULL) {
313                         exit(2); /* out of memory */
314                 }
315         }
316
317         if (file) {
318                 if ((fptr = fopen(file, "r")) == NULL) {
319                         (void) fprintf(stderr,
```



```
320                                 gettext("fgrep: can't open %s\n"), file);
321                         retcode = 2;
322                         return;
323                 }
324         } else {
325                 file = "<stdin>";
326                 fptr = stdin;
327                 file = STDIN_FILENAME;
328 #endif /* ! codereview */
329         }
330         ccount = 0;
331         failed = 0;
332         lnum = 1;
333         tln = 0;
334         blkno = 0;
335         p = buf;
336         nlp = p;
337         c = w;
338         for (;;) {
339                 if (c == 0)
340                         break;
341                 if (ccount <= 0) {
342                         if (p >= &buf[fw_lBufsiz + BUFSIZ]) {
343                                 if (nlp == buf) {
344                                         /* increase the buffer size */
345                                         fw_lBufsiz += BUFSIZ;
346                                         if ((buf = realloc(buf,
347                                                 fw_lBufsiz + BUFSIZ)) == NULL) {
348                                                 exit(2); /* out of memory */
349                                         }
350                                         nlp = buf;
351                                         p = &buf[fw_lBufsiz];
352                                 } else {
353                                         /* shift the buffer down */
354                                         (void) memmove(buf, nlp,
355                                                 &buf[fw_lBufsiz + BUFSIZ]
356                                                 - nlp);
357                                         p -= nlp - buf;
358                                         nlp = buf;
359                                 }
360
361                         }
362                         if (p > &buf[fw_lBufsiz]) {
363                                 if ((ccount = fread(p, sizeof (char),
364                                         &buf[fw_lBufsiz + BUFSIZ] - p, fptr))
365                                         <= 0)
366                                         break;
367                         } else if ((ccount = fread(p, sizeof (char),
368                                         BUFSIZ, fptr)) <= 0)
369                                         break;
370                         blkno += (long long)ccount;
371                 }
372                 GETONE(lc, p);
373 nstate:
374                 if (same(c->inp, lc)) {
375                         c = c->nst;
376                 } else if (c->link != 0) {
377                         c = c->link;
378                         goto nstate;
379                 } else {
380                         c = c->fail;
381                         failed = 1;
382                         if (c == 0) {
383                                 c = w;
384 istate:
                                        if (same(c->inp, lc)) {
```

```
385                                     c = c->nst;
386                             } else if (c->link != 0) {
387                                     c = c->link;
388                                     goto istate;
389                             }
390                     } else
391                             goto nstate;
392             }

394             if (c == 0)
395                     break;

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



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

_____unchanged_portion_omitted_

```
**********************************************************
   10506 Thu May 23 21:03:50 2013
new/usr/src/cmd/grep/grep.c
3737 grep does not support -H option
**********************************************************
______unchanged_portion_omitted_
```

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

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

 84 static int      temp;
 85 static long long        lnum;
 86 static char     *linebuf;
 87 static char     *prntbuf = NULL;
 88 static long     fw_lPrntBufLen = 0;
 89 static int      nflag;
 90 static int      bflag;
 91 static int      lflag;
 92 static int      cflag;
 93 static int      rflag;
 94 static int      Rflag;
 95 static int      vflag;
 96 static int      sflag;
 97 static int      iflag;
 98 static int      wflag;
 99 static int      hflag;
100 static int      Hflag;
101 #endif /* ! codereview */
102 static int      qflag;
103 static int      errflg;
104 static int      nfile;
105 static long long        tln;
106 static int      nsucc;
107 static int      outfn = 0;
108 static int      nlflag;
109 static char     *ptr, *ptrend;
110 static char     *expbuf;

112 static void     execute(const char *, int);
113 static void     regerr(int);
114 static void     prepare(const char *);
115 static int      recursive(const char *, const struct stat *, int, struct FTW *);
116 static int      succeed(const char *);

118 int
119 main(int argc, char **argv)
120 {
121         int     c;
122         char    *arg;
123         extern int      optind;

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

131         while ((c = getopt(argc, argv, "hHqblcnRrsviyw")) != -1)
 76         while ((c = getopt(argc, argv, "hqblcnRrsviyw")) != -1)
132                 switch (c) {
133                 /* based on options order h or H is set as in GNU grep */
```



```
134 #endif /* ! codereview */
135                 case 'h':
136                         hflag++;
137                         Hflag = 0; /* h excludes H */
138                         break;
139                 case 'H':
140                         if (!lflag) /* H is excluded by l */
141                                 Hflag++;
142                         hflag = 0; /* H excludes h */
143 #endif /* ! codereview */
144                         break;
145                 case 'q':       /* POSIX: quiet: status only */
146                         qflag++;
147                         break;
148                 case 'v':
149                         vflag++;
150                         break;
151                 case 'c':
152                         cflag++;
153                         break;
154                 case 'n':
155                         nflag++;
156                         break;
157                 case 'R':
158                         Rflag++;
159                         /* FALLTHROUGH */
160                 case 'r':
161                         rflag++;
162                         break;
163                 case 'b':
164                         bflag++;
165                         break;
166                 case 's':
167                         sflag++;
168                         break;
169                 case 'l':
170                         lflag++;
171                         Hflag = 0; /* l excludes H */
172 #endif /* ! codereview */
173                         break;
174                 case 'y':
175                 case 'i':
176                         iflag++;
177                         break;
178                 case 'w':
179                         wflag++;
180                         break;
181                 case '?':
182                         errflg++;
183                 }

185         if (errflg || (optind >= argc)) {
186                 errmsg("Usage: grep [-c|-l|-q] [-r|-R] -hHbnsviw "
 78                 errmsg("Usage: grep [-c|-l|-q] [-r|-R] -hbnsviw "
187                     "pattern file . . .\n",
188                     (char *)NULL);
189                 exit(2);
190         }

192         argv = &argv[optind];
193         argc -= optind;
194         nfile = argc - 1;

196         if (strrchr(*argv, '\n') != NULL)
197                 regerr(41);
```

```
199         if (iflag) {
200                 for (arg = *argv; *arg != NULL; ++arg)
201                         *arg = (char)tolower((int)((unsigned char)*arg));
202         }

204         if (wflag) {
205                 unsigned int    wordlen;
206                 char            *wordbuf;

208                 wordlen = strlen(*argv) + 5; /* '\\' '<' *argv '\\' '>' '\0' */
209                 if ((wordbuf = malloc(wordlen)) == NULL) {
210                         errmsg("grep: Out of memory for word\n", (char *)NULL);
211                         exit(2);
212                 }

214                 (void) strcpy(wordbuf, "\\<");
215                 (void) strcat(wordbuf, *argv);
216                 (void) strcat(wordbuf, "\\>");
217                 *argv = wordbuf;
218         }

220         expbuf = compile(*argv, (char *)0, (char *)0);
221         if (regerrno)
222                 regerr(regerrno);

224         if (--argc == 0)
225                 execute(NULL, 0);
226         else
227                 while (argc-- > 0)
228                         prepare(*++argv);

230         return (nsucc == 2 ? 2 : (nsucc == 0 ? 1 : 0));
231 }
```

*unchanged_portion_omitted_*

```
297 static void
298 execute(const char *file, int base)
299 {
300         char    *lbuf, *p;
301         long    count;
302         long    offset = 0;
303         char    *next_ptr = NULL;
304         long    next_count = 0;

306         tln = 0;

308         if (prntbuf == NULL) {
309                 fw_lPrntBufLen = GBUFSIZ + 1;
310                 if ((prntbuf = malloc(fw_lPrntBufLen)) == NULL) {
311                         exit(2); /* out of memory - BAIL */
312                 }
313                 if ((linebuf = malloc(fw_lPrntBufLen)) == NULL) {
314                         exit(2); /* out of memory - BAIL */
315                 }
316         }

318         if (file == NULL) {
210         if (file == NULL)
319                 temp = 0;
320                 file = STDIN_FILENAME;
321         } else if ((temp = open(file + base, O_RDONLY)) == -1) {
212         else if ((temp = open(file + base, O_RDONLY)) == -1) {
322                 if (!sflag)
323                         errmsg("grep: can't open %s\n", file);
324                 nsucc = 2;
325                 return;
```



```
326         }

328         /* read in first block of bytes */
329         if ((count = read(temp, prntbuf, GBUFSIZ)) <= 0) {
330                 (void) close(temp);

332                 if (cflag && !qflag) {
333                         if (Hflag || (nfile > 1 && !hflag))
224                         if (nfile > 1 && !hflag && file)
334                                 (void) fprintf(stdout, "%s:", file);
335                         if (!rflag)
336                         (void) fprintf(stdout, "%lld\n", tln);
337                 }
338                 return;
339         }

341         lnum = 0;
342         ptr = prntbuf;
343         for (;;) {
344                 /* look for next newline */
345                 if ((ptrend = memchr(ptr + offset, '\n', count)) == NULL) {
346                         offset += count;

348                         /*
349                          * shift unused data to the beginning of the buffer
350                          */
351                         if (ptr > prntbuf) {
352                                 (void) memmove(prntbuf, ptr, offset);
353                                 ptr = prntbuf;
354                         }

356                         /*
357                          * re-allocate a larger buffer if this one is full
358                          */
359                         if (offset + GBUFSIZ > fw_lPrntBufLen) {
360                                 /*
361                                  * allocate a new buffer and preserve the
362                                  * contents...
363                                  */
364                                 fw_lPrntBufLen += GBUFSIZ;
365                                 if ((prntbuf = realloc(prntbuf,
366                                     fw_lPrntBufLen)) == NULL)
367                                         exit(2);

369                                 /*
370                                  * set up a bigger linebuffer (this is only used
371                                  * for case insensitive operations). Contents do
372                                  * not have to be preserved.
373                                  */
374                                 free(linebuf);
375                                 if ((linebuf = malloc(fw_lPrntBufLen)) == NULL)
376                                         exit(2);

378                                 ptr = prntbuf;
379                         }

381                         p = prntbuf + offset;
382                         if ((count = read(temp, p, GBUFSIZ)) > 0)
383                                 continue;

385                         if (offset == 0)
386                                 /* end of file already reached */
387                                 break;

389                         /* last line of file has no newline */
390                         ptrend = ptr + offset;
```

```
391                     nlflag = 0;
392             } else {
393                     next_ptr = ptrend + 1;
394                     next_count = offset + count - (next_ptr - ptr);
395                     nlflag = 1;
396             }
397             lnum++;
398             *ptrend = '\0';

400             if (iflag) {
401                     /*
402                      * Make a lower case copy of the record
403                      */
404                     p = ptr;
405                     for (lbuf = linebuf; p < ptrend; )
406                             *lbuf++ = (char)tolower((int)
407                                 (unsigned char)*p++);
408                     *lbuf = '\0';
409                     lbuf = linebuf;
410             } else
411                     /*
412                      * Use record as is
413                      */
414                     lbuf = ptr;

416             /* lflag only once */
417             if ((step(lbuf, expbuf) ^ vflag) && succeed(file) == 1)
418                     break;

420             if (!nlflag)
421                     break;

423             ptr = next_ptr;
424             count = next_count;
425             offset = 0;
426     }
427     (void) close(temp);

429     if (cflag && !qflag) {
430             if (Hflag || (!hflag && ((nfile > 1) ||
431                 (rflag && outfn))))
321             if (!hflag && file && (nfile > 1 ||
322                 (rflag && outfn)))
432                     (void) fprintf(stdout, "%s:", file);
433             (void) fprintf(stdout, "%lld\n", tln);
434     }
435 }

437 static int
438 succeed(const char *f)
439 {
440     int nchars;
441     nsucc = (nsucc == 2) ? 2 : 1;

334     if (f == NULL)
335             f = "<stdin>";

443     if (qflag) {
444             /* no need to continue */
445             return (1);
446     }

448     if (cflag) {
449             tln++;
450             return (0);
451     }
```



```
453     if (lflag) {
454             (void) fprintf(stdout, "%s\n", f);
455             return (1);
456     }

458     if (Hflag || (!hflag && (nfile > 1 || (rflag && outfn)))) {
352     if (!hflag && (nfile > 1 || (rflag && outfn))) {
459             /* print filename */
460             (void) fprintf(stdout, "%s:", f);
461     }

463     if (bflag)
464             /* print block number */
465             (void) fprintf(stdout, "%lld:", (offset_t)
466                 ((lseek(temp, (off_t)0, SEEK_CUR) - 1) / BLKSIZE));

468     if (nflag)
469             /* print line number */
470             (void) fprintf(stdout, "%lld:", lnum);

472     if (nlflag) {
473             /* newline at end of line */
474             *ptrend = '\n';
475             nchars = ptrend - ptr + 1;
476     } else {
477             /* don't write sentinel \0 */
478             nchars = ptrend - ptr;
479     }

481     (void) fwrite(ptr, 1, nchars, stdout);
482     return (0);
483 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   28311 Thu May 23 21:03:50 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. */

#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
 */
int
main(int argc, char **argv)
{
        char    *ap;
        int     c;
```

```
128         int     fflag = 0;
129         int     i, n_pattern = 0, n_file = 0;
130         char    **pattern_list = NULL;
131         char    **file_list = NULL;

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

139         /*
140          * true if this is running on the multibyte locale
141          */
142         mblocale = (MB_CUR_MAX > 1);
143         /*
144          * Skip leading slashes
145          */
146         cmdname = argv[0];
147         if (ap = strrchr(cmdname, '/'))
148                 cmdname = ap + 1;

150         ap = cmdname;
151         /*
152          * Detect egrep/fgrep via command name, map to -E and -F options.
153          */
154         if (*ap == 'e' || *ap == 'E') {
155                 regflags |= REG_EXTENDED;
156                 egrep++;
157         } else {
158                 if (*ap == 'f' || *ap == 'F') {
159                         fgrep++;
160                 }
161         }

163         while ((c = getopt(argc, argv, "vwchHilnrbse:f:qxEFIR")) != EOF) {
 57         while ((c = getopt(argc, argv, "vwchilnrbse:f:qxEFIR")) != EOF) {
164                 switch (c) {
165                 case 'v':       /* POSIX: negate matches */
166                         nvflag = 0;
167                         break;

169                 case 'c':       /* POSIX: write count */
170                         cflag++;
171                         break;

173                 case 'i':       /* POSIX: ignore case */
174                         iflag++;
175                         regflags |= REG_ICASE;
176                         break;

178                 case 'l':       /* POSIX: Write filenames only */
179                         lflag++;
180                         break;

182                 case 'n':       /* POSIX: Write line numbers */
183                         nflag++;
184                         break;

186                 case 'r':       /* Solaris: search recursively */
187                         rflag++;
188                         break;

190                 case 'b':       /* Solaris: Write file block numbers */
191                         bflag++;
192                         break;
```



```
194                 case 's':       /* POSIX: No error msgs for files */
195                         sflag++;
196                         break;

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

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

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

238                 case 'q':       /* POSIX: quiet: status only */
239                         qflag++;
240                         break;

242                 case 'w':       /* Solaris: treat pattern as word */
243                         wflag++;
244                         break;

246                 case 'x':       /* POSIX: full line matches */
247                         xflag++;
248                         regflags |= REG_ANCHOR;
249                         break;

251                 case 'E':       /* POSIX: Extended RE's */
252                         regflags |= REG_EXTENDED;
253                         Eflag++;
254                         break;

256                 case 'F':       /* POSIX: strings, not RE's */
257                         Fflag++;
258                         break;
```

```
260                 case 'R':       /* Solaris: like rflag, but follow symlinks */
261                         Rflag++;
262                         rflag++;
263                         break;

265                 default:
266                         usage();
267                 }
268         }
269         /*
270          * If we're invoked as egrep or fgrep we need to do some checks
271          */

273         if (egrep || fgrep) {
274                 /*
275                  * Use of -E or -F with egrep or fgrep is illegal
276                  */
277                 if (Eflag || Fflag)
278                         usage();
279                 /*
280                  * Don't allow use of wflag with egrep / fgrep
281                  */
282                 if (wflag)
283                         usage();
284                 /*
285                  * For Solaris the -s flag is equivalent to XCU -q
286                  */
287                 if (sflag)
288                         qflag++;
289                 /*
290                  * done with above checks - set the appropriate flags
291                  */
292                 if (egrep)
293                         Eflag++;
294                 else                    /* Else fgrep */
295                         Fflag++;
296         }

298         if (wflag && (Eflag || Fflag)) {
299                 /*
300                  * -w cannot be specified with grep -F
301                  */
302                 usage();
303         }

305         /*
306          * -E and -F flags are mutually exclusive - check for this
307          */
308         if (Eflag && Fflag)
309                 usage();

311         /*
312          * -l overrides -H like in GNU grep
313          */
314         if (lflag)
315                 Hflag = 0;

317         /*
318 #endif /* ! codereview */
319          * -c, -l and -q flags are mutually exclusive
320          * We have -c override -l like in Solaris.
321          * -q overrides -l & -c programmatically in grep() function.
322          */
323         if (cflag && lflag)
324                 lflag = 0;
```



```
326         argv += optind - 1;
327         argc -= optind - 1;

329         /*
330          * Now handling -e and -f option
331          */
332         if (pattern_list) {
333                 for (i = 0; i < n_pattern; i++) {
334                         addpattern(pattern_list[i]);
335                 }
336                 free(pattern_list);
337         }
338         if (file_list) {
339                 for (i = 0; i < n_file; i++) {
340                         addfile(file_list[i]);
341                 }
342                 free(file_list);
343         }

345         /*
346          * No -e or -f?  Make sure there is one more arg, use it as the pattern.
347          */
348         if (patterns == NULL && !fflag) {
349                 if (argc < 2)
350                         usage();
351                 addpattern(argv[1]);
352                 argc--;
353                 argv++;
354         }

356         /*
357          * If -x flag is not specified or -i flag is specified
358          * with fgrep in a multibyte locale, need to use
359          * the wide character APIs.  Otherwise, byte-oriented
360          * process will be done.
361          */
362         use_wchar = Fflag && mblocale && (!xflag || iflag);

364         /*
365          * Compile Patterns and also decide if BMG can be used
366          */
367         fixpatterns();

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

383         (void) fflush(stdout);

385         if (errors)
386                 return (2);
387         return (matched ? 0 : 1);
388 }
```

*_____unchanged_portion_omitted_*

```
789 /*
790  * Do grep on a single file.
791  * Return true in any lines matched.
792  *
793  * We have two strategies:
794  * The fast one is used when we have a single pattern with
795  * a string known to occur in the pattern. We can then
796  * do a BMG match on the whole buffer.
797  * This is an order of magnitude faster.
798  * Otherwise we split the buffer into lines,
799  * and check for a match on each line.
800  */
801 static int
802 grep(int fd, const char *fn)
803 {
804         PATTERN *pp;
805         off_t   data_len;       /* length of the data chunk */
806         off_t   line_len;       /* length of the current line */
807         off_t   line_offset;    /* current line's offset from the beginning */
808         long long       lineno;
809         long long       matches = 0;    /* Number of matching lines */
810         int     newlinep;       /* 0 if the last line of file has no newline */
811         char    *ptr, *ptrend;


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

817         pp = patterns;

819         if (use_bmg) {
820                 bmgcomp(pp->pattern, strlen(pp->pattern));
821         }

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

833         if (prntbuf == NULL) {
834                 prntbuflen = BUFSIZE;
835                 if ((prntbuf = malloc(prntbuflen + 1)) == NULL) {
836                         (void) fprintf(stderr, gettext("%s: out of memory\n"),
837                             cmdname);
838                         exit(2);
839                 }
840         }

842         line_offset = 0;
843         lineno = 0;
844         newlinep = 1;
845         data_len = 0;
846         for (; ; ) {
847                 long    count;
848                 off_t   offset = 0;

850                 if (data_len == 0) {
851                         /*
852                          * If no data in the buffer, reset ptr
853                          */
```



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

892                 /*
893                  * Look for newline in the chunk
894                  * between ptr + offset and ptr + data_len - offset.
895                  */
896                 ptrend = find_nl(ptr + offset, data_len - offset);
897                 if (ptrend == NULL) {
898                         /* no newline found in this chunk */
899                         if (ptr > prntbuf) {
900                                 /*
901                                  * Move remaining data to the beginning
902                                  * of the buffer.
903                                  * Remaining data lie from ptr for
904                                  * data_len bytes.
905                                  */
906                                 (void) memmove(prntbuf, ptr, data_len);
907                         }
908                         if (data_len == prntbuflen) {
909                                 /*
910                                  * No enough room in the buffer
911                                  */
912                                 prntbuflen += BUFSIZE;
913                                 prntbuf = realloc(prntbuf, prntbuflen + 1);
914                                 if (prntbuf == NULL) {
915                                         (void) fprintf(stderr,
916                                             gettext("%s: out of memory\n"),
917                                             cmdname);
918                                         exit(2);
919                                 }
```

```
920                         }
921                         ptr = prntbuf;
922                         /* read the next input */
923                         continue;
924                 }
925 L_start_process:

927                 /*
928                  * Beginning of the chunk:      ptr
929                  * End of the chunk:            ptr + data_len
930                  * Beginning of the line:       ptr
931                  * End of the line:             ptrend
932                  */

934                 if (use_bmg) {
935                         /*
936                          * Use Boyer-Moore-Gosper algorithm to find out if
937                          * this chunk (not this line) contains the specified
938                          * pattern.  If not, restart from the last line
939                          * of this chunk.
940                          */
941                         char    *bline;
942                         bline = bmgexec(ptr, ptr + data_len);
943                         if (bline == NULL) {
944                                 /*
945                                  * No pattern found in this chunk.
946                                  * Need to find the last line
947                                  * in this chunk.
948                                  */
949                                 ptrend = rfind_nl(ptr, data_len);

951                                 /*
952                                  * When this chunk does not contain newline,
953                                  * ptrend becomes NULL, which should happen
954                                  * when the last line of file does not end
955                                  * with a newline.  At such a point,
956                                  * newlinep should have been set to 0.
957                                  * Therefore, just after jumping to
958                                  * L_skip_line, the main for-loop quits,
959                                  * and the line_len value won't be
960                                  * used.
961                                  */
962                                 line_len = ptrend - ptr;
963                                 goto L_skip_line;
964                         }
965                         if (bline > ptrend) {
966                                 /*
967                                  * Pattern found not in the first line
968                                  * of this chunk.
969                                  * Discard the first line.
970                                  */
971                                 line_len = ptrend - ptr;
972                                 goto L_skip_line;
973                         }
974                         /*
975                          * Pattern found in the first line of this chunk.
976                          * Using this result.
977                          */
978                         *ptrend = '\0';
979                         line_len = ptrend - ptr;

981                         /*
982                          * before jumping to L_next_line,
983                          * need to handle xflag if specified
984                          */
985                         if (xflag && (line_len != bmglen ||
```



```
986                             strcmp(bmgpat, ptr) != 0)) {
987                                 /* didn't match */
988                                 pp = NULL;
989                         } else {
990                                 pp = patterns; /* to make it happen */
991                         }
992                         goto L_next_line;
993                 }
994                 lineno++;
995                 /*
996                  * Line starts from ptr and ends at ptrend.
997                  * line_len will be the length of the line.
998                  */
999                 *ptrend = '\0';
1000                line_len = ptrend - ptr;

1002                /*
1003                 * From now, the process will be performed based
1004                 * on the line from ptr to ptrend.
1005                 */
1006                if (use_wchar) {
1007                        size_t  len;

1009                        if (line_len >= outbuflen) {
1010                                outbuflen = line_len + 1;
1011                                outline = realloc(outline,
1012                                    sizeof (wchar_t) * outbuflen);
1013                                if (outline == NULL) {
1014                                        (void) fprintf(stderr,
1015                                            gettext("%s: out of memory\n"),
1016                                            cmdname);
1017                                        exit(2);
1018                                }
1019                        }

1021                        len = mbstowcs(outline, ptr, line_len);
1022                        if (len == (size_t)-1) {
1023                                (void) fprintf(stderr, gettext(
1024        "%s: input file \"%s\": line %lld: invalid multibyte character\n"),
1025                                    cmdname, fn, lineno);
1026                                /* never match a line with invalid sequence */
1027                                goto L_skip_line;
1028                        }
1029                        outline[len] = L'\0';

1031                        if (iflag) {
1032                                wchar_t *cp;
1033                                for (cp = outline; *cp != '\0'; cp++) {
1034                                        *cp = towlower((wint_t)*cp);
1035                                }
1036                        }

1038                        if (xflag) {
1039                                for (pp = patterns; pp; pp = pp->next) {
1040                                        if (outline[0] == pp->wpattern[0] &&
1041                                            wcscmp(outline,
1042                                            pp->wpattern) == 0) {
1043                                                /* matched */
1044                                                break;
1045                                        }
1046                                }
1047                        } else {
1048                                for (pp = patterns; pp; pp = pp->next) {
1049                                        if (wcswcs(outline, pp->wpattern)
1050                                            != NULL) {
1051                                                /* matched */
```

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

1086                            rv = regexec(&pp->re, ptr, 0, NULL, 0);
1087                            if (rv == REG_OK) {
1088                                    /* matched */
1089                                    break;
1090                            }

1092                            switch (rv) {
1093                            case REG_NOMATCH:
1094                                    break;
1095                            case REG_ECHAR:
1096                                    (void) fprintf(stderr, gettext(
1097            "%s: input file \"%s\": line %lld: invalid multibyte character\n"),
1098                                        cmdname, fn, lineno);
1099                                    break;
1100                            default:
1101                                    (void) regerror(rv, &pp->re, errstr,
1102                                        sizeof (errstr));
1103                                    (void) fprintf(stderr, gettext(
1104            "%s: input file \"%s\": line %lld: %s\n"),
1105                                        cmdname, fn, lineno, errstr);
1106                                    exit(2);
1107                            }
1108                    }
1109            }

1111 L_next_line:
1112                 /*
1113                  * Here, if pp points to non-NULL, something has been matched
1114                  * to the pattern.
1115                  */
1116                 if (nvflag == (pp != NULL)) {
1117                         matches++;
```



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

1158                 data_len -= line_len + 1;
1159                 line_offset += line_len + 1;
1160                 ptr = ptrend + 1;
1161         }

1163         if (cflag) {
1164                 if (Hflag || outfn) {
 911                 if (outfn) {
1165                         (void) printf("%s:", fn);
1166                 }
1167                 if (!qflag) {
1168                         (void) printf("%lld\n", matches);
1169                 }
1170         }
1171         return (matches != 0);
1172 }

1174 /*
1175  * usage message for grep
1176  */
1177 static void
1178 usage(void)
1179 {
1180         if (egrep || fgrep) {
1181                 (void) fprintf(stderr, gettext("Usage:\t%s"), cmdname);
```

```
1182                (void) fprintf(stderr,
1183                    gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvx] "
 930                    gettext(" [-c|-l|-q] [-r|-R] [-bhinsvx] "
1184                    "pattern_list [file ...]\n"));

1186                (void) fprintf(stderr, "\t%s", cmdname);
1187                (void) fprintf(stderr,
1188                    gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvx] "
 935                    gettext(" [-c|-l|-q] [-r|-R] [-bhinsvx] "
1189                    "[-e pattern_list]... "
1190                    "[-f pattern_file]... [file...]\n"));
1191        } else {
1192                (void) fprintf(stderr, gettext("Usage:\t%s"), cmdname);
1193                (void) fprintf(stderr,
1194                    gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvwx] "
 941                    gettext(" [-c|-l|-q] [-r|-R] [-bhinsvwx] "
1195                    "pattern_list [file ...]\n"));

1197                (void) fprintf(stderr, "\t%s", cmdname);
1198                (void) fprintf(stderr,
1199                    gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvwx] "
 946                    gettext(" [-c|-l|-q] [-r|-R] [-bhinsvwx] "
1200                    "[-e pattern_list]... "
1201                    "[-f pattern_file]... [file...]\n"));

1203                (void) fprintf(stderr, "\t%s", cmdname);
1204                (void) fprintf(stderr,
1205                    gettext(" -E [-c|-l|-q] [-r|-R] [-bhHinsvx] "
 952                    gettext(" -E [-c|-l|-q] [-r|-R] [-bhinsvx] "
1206                    "pattern_list [file ...]\n"));

1208                (void) fprintf(stderr, "\t%s", cmdname);
1209                (void) fprintf(stderr,
1210                    gettext(" -E [-c|-l|-q] [-r|-R] [-bhHinsvx] "
 957                    gettext(" -E [-c|-l|-q] [-r|-R] [-bhinsvx] "
1211                    "[-e pattern_list]... "
1212                    "[-f pattern_file]... [file...]\n"));

1214                (void) fprintf(stderr, "\t%s", cmdname);
1215                (void) fprintf(stderr,
1216                    gettext(" -F [-c|-l|-q] [-r|-R] [-bhHinsvx] "
 963                    gettext(" -F [-c|-l|-q] [-r|-R] [-bhinsvx] "
1217                    "pattern_list [file ...]\n"));

1219                (void) fprintf(stderr, "\t%s", cmdname);
1220                (void) fprintf(stderr,
1221                    gettext(" -F [-c|-l|-q] [-bhHinsvx] [-e pattern_list]... "
 968                    gettext(" -F [-c|-l|-q] [-bhinsvx] [-e pattern_list]... "
1222                    "[-f pattern_file]... [file...]\n"));
1223        }
1224        exit(2);
1225        /* NOTREACHED */
1226 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    9118 Thu May 23 21:03:51 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 23 21:03:51 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 23 21:03:51 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.
```