```
**********************************************************
   26110 Fri Sep 13 10:33:20 2013
new/usr/src/cmd/egrep/egrep.y
3737 grep does not support -H option
3759 egrep(1) and fgrep(1) -s flag does not hide -c output
Reviewed by: Albert Lee <trisk@nexenta.com>
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
**********************************************************
```

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

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

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

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

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

%{
#include <stdio.h>
#include <ctype.h>
```



```
#include <memory.h>
#include <wchar.h>
#include <wctype.h>
#include <widec.h>
#include <stdlib.h>
#include <limits.h>
#include <locale.h>

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

#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;
int     hflag;
int     iflag;
int     lflag;
int     nflag;
int     qflag;
int     sflag;
```

```
121 int     vflag;
122 int     nfile;
123 long long blkno;
124 long long tln;
125 int     nsucc;
126 int     badbotch;
127 extern  char *optarg;
128 extern  int optind;

130 int     f;
131 FILE    *expfile;
132 %}

134 %%
135 s:      t
136                 {
137                   unary(FINAL, $1);
138                   line--;
139                 }
140         ;
141 t:      b r
142                 { $$ = node(CAT, $1, $2); }
143         | OR b r OR
144                 { $$ = node(CAT, $2, $3); }
145         | OR b r
146                 { $$ = node(CAT, $2, $3); }
147         | b r OR
148                 { $$ = node(CAT, $1, $2); }
149         ;
150 b:
151                 { /* if(multibyte)
152                         $$ = mdotenter();
153                   else */
154                         $$ = enter(DOT);
155                   $$ = unary(STAR, $$);
156                 }
157         ;
158 r:      CHAR
159                 { $$ = iflag && isalpha($1) ?
160                 node(OR, enter(tolower($1)), enter(toupper($1))) : enter($1); }
161         | MCHAR
162                 { $$ = (iflag && iswalpha(lyylval)) ?
163                 node(OR, mchar(towlower(lyylval)), mchar(towupper(lyylval))) :
164                 mchar(lyylval); }
165         | DOT
166                 { if(multibyte)
167                         $$ = mdotenter();
168                   else
169                         $$ = enter(DOT);
170                 }
171         | CCL
172                 { $$ = cclenter(CCL); }
173         | NCCL
174                 { $$ = cclenter(NCCL); }
175         | MCCL
176                 { $$ = ccl(CCL); }
177         | NMCCL
178                 { $$ = ccl(NCCL); }
179         ;

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



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

196 %%
197 void    add(int *, int);
198 void    clearg(void);
199 void    execute(char *);
200 void    follow(int);
201 int     mgetc(void);
202 void    synerror(void);


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

*unchanged_portion_omitted_*

```
650 #define USAGE "[ -bchHilnsqv ] [ -e exp ] [ -f file ] [ strings ] [ file ] ..."
647 #define USAGE "[ -bchilnsv ] [ -e exp ] [ -f file ] [ strings ] [ file ] ..."

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

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

666         while((c = getopt(argc, argv, "ybcie:f:Hhlnvsq")) != -1)
663         while((c = getopt(argc, argv, "ybcie:f:hlnvs")) != -1)
667                 switch(c) {

669                 case 'b':
670                         bflag++;
671                         continue;

673                 case 'c':
674                         cflag++;
675                         continue;

677                 case 'e':
678                         eflag++;
679                         input = optarg;
680                         continue;

682                 case 'f':
683                         fflag++;
684                         expfile = fopen(optarg, "r");
685                         if(expfile == NULL) {
686                                 fprintf(stderr,
687                                   gettext("egrep: can't open %s\n"), optarg);
```

```
688                                 exit(2);
689                         }
690                         continue;

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

698                 case 'h':
699                         hflag++;
700                         Hflag = 0; /* h excludes H */
701                         continue;

703                 case 'y':
704                 case 'i':
705                         iflag++;
706                         continue;

708                 case 'l':
709                         lflag++;
710                         Hflag = 0; /* l excludes H */
711                         continue;

713                 case 'n':
714                         nflag++;
715                         continue;

717                 case 'q':
718                 case 's': /* Solaris: legacy option */
719                         qflag++;
706                 case 's':
707                         sflag++;
720                         continue;

722                 case 'v':
723                         vflag++;
724                         continue;

726                 case '?':
727                         errflag++;
728                 }
729         if (errflag || ((argc <= 0) && !fflag && !eflag)) {
730                 fprintf(stderr, gettext("usage: egrep %s\n"), gettext(USAGE));
731                 exit(2);
732         }
733         if(!eflag && !fflag) {
734                 input = argv[optind];
735                 optind++;
736         }

738         argc -= optind;
739         argv = &argv[optind];
740
741         /* allocate initial space for arrays */
742         if((name = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
743                 overflo();
744         if((left = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
745                 overflo();
746         if((right = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
747                 overflo();
748         if((parent = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
749                 overflo();
750         if((foll = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
751                 overflo();
```



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

771         cfoll(line-1);
772         cgotofn();
773         nfile = argc;
774         if (argc<=0) {
775                 execute(0);
776         }
777         else while (--argc >= 0) {
778                 if (reinit == 1) clearg();
779                 execute(*argv++);
780         }
781         return (badbotch ? 2 : nsucc==0);
782 }

784 void
785 execute(char *file)
786 {
787         char *p;
788         int cstat;
789         wchar_t c;
790         int t;
791         long count;
792         long count1, count2;
793         long nchars;
794         int succ;
795         char *ptr, *ptrend, *lastptr;
796         char *buf;
797         long lBufSiz;
798         FILE *f;
799         int nlflag;

801         lBufSiz = EBUFSIZ;
802         if ((buf = malloc (lBufSiz + EBUFSIZ)) == NULL) {
803                 exit (2); /* out of memory - BAIL */
804         }

806         if (file) {
807                 if ((f = fopen(file, "r")) == NULL) {
808                         fprintf(stderr,
809                                 gettext("egrep: can't open %s\n"), file);
810                         badbotch=1;
811                         return;
812                 }
813         } else {
802                 file = "<stdin>";
814                 f = stdin;
815                 file = STDIN_FILENAME;
816         }
```

```
817         lnum = 1;
818         tln = 0;
819         if((count = read(fileno(f), buf, EBUFSIZ)) <= 0) {
820                 fclose(f);

822                 if (cflag && !qflag) {
823                         if (Hflag || (nfile > 1 && !hflag))
810                 if (cflag) {
811                         if (nfile>1 && !hflag)
824                                 fprintf(stdout, "%s:", file);
825                         fprintf(stdout, "%lld\n", tln);
826                 }
827                 return;
828         }

830         blkno = count;
831         ptr = buf;
832         for(;;) {
833                 if((ptrend = memchr(ptr, '\n', buf + count - ptr)) == NULL) {
834                         /*
835                                 move the unused partial record to the head of th
836                         */
837                         if (ptr > buf) {
838                                 count = buf + count - ptr;
839                                 memmove (buf, ptr, count);
840                                 ptr = buf;
841                         }

843                         /*
844                                 Get a bigger buffer if this one is full
845                         */
846                         if(count > lBufSiz) {
847                                 /*
848                                         expand the buffer
849                                 */
850                                 lBufSiz += EBUFSIZ;
851                                 if ((buf = realloc (buf, lBufSiz + EBUFSIZ)) ==
852                                         exit (2); /* out of memory - BAIL */
853                                 }

855                                 ptr = buf;
856                         }

858                         p = buf + count;
859                         if((count1 = read(fileno(f), p, EBUFSIZ)) > 0) {
860                                 count += count1;
861                                 blkno += count1;
862                                 continue;
863                         }
864                         ptrend = ptr + count;
865                         nlflag = 0;
866                 } else
867                         nlflag = 1;
868                 *ptrend = '\n';
869                 p = ptr;
870                 lastptr = ptr;
871                 cstat = istat;
872                 succ = 0;
873                 for(;;) {
874                         if(out[cstat]) {
875                                 if(multibyte && p > ptr) {
876                                         wchar_t wchar;
877                                         int length;
878                                         char *endptr = p;
879                                         p = lastptr;
880                                         while(p < endptr) {
```



```
881                                             length = mbtowc(&wchar, p, MB_LE
882                                             if(length <= 1)
883                                                     p++;
884                                             else
885                                                     p += length;
886                                     }
887                                     if(p == endptr) {
888                                             succ = !vflag;
889                                             break;
890                                     }
891                                     cstat = 1;
892                                     length = mbtowc(&wchar, lastptr, MB_LEN_
893                                     if(length <= 1)
894                                             lastptr++;
895                                     else
896                                             lastptr += length;
897                                     p = lastptr;
898                                     continue;
899                             }
900                             succ = !vflag;
901                             break;
902                     }
903                     c = (unsigned char)*p++;
904                     if ((t = gotofn[cstat][c]) == 0)
905                             cstat = nxtst(cstat, c);
906                     else
907                             cstat = t;
908                     if(c == RIGHT) {
909                             if(out[cstat]) {
910                                     succ = !vflag;
911                                     break;
912                             }
913                             succ = vflag;
914                             break;
915                     }
916             }
917             if (succ) {
905             if(succ) {
918                     nsucc = 1;
919                     if (lflag || qflag) {
920                             if (!qflag)
921                                     (void) printf("%s\n", file);
907                     if (cflag) tln++;
908                     else if (sflag)
909                             ;       /* ugh */
910                     else if (lflag) {
911                             printf("%s\n", file);
922                             fclose(f);
923                             return;
924                     }
925                     if (cflag) {
926                             tln++;
927                     } else {
928                             if (Hflag || (nfile > 1 && !hflag))
929                                     printf("%s:", file);
915                     else {
916                             if (nfile > 1 && !hflag)
917                                     printf(gettext("%s:"), file);
930                             if (bflag) {
931                                     nchars = blkno - (buf + count - ptrend)
932                                     if(nlflag)
933                                             nchars++;
934                                     printf("%lld:", nchars/BLKSIZE);
935                             }
936                             if (nflag)
937                                     printf("%lld:", lnum);
```

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

*_____unchanged_portion_omitted_*

```
**********************************************************
   14491 Fri Sep 13 10:33:20 2013
new/usr/src/cmd/fgrep/fgrep.c
3737 grep does not support -H option
3759 egrep(1) and fgrep(1) -s flag does not hide -c output
Reviewed by: Albert Lee <trisk@nexenta.com>
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

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

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

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

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

#include <getwidth.h>
```



```
eucwidth_t WW;
#define WIDTH1  WW._eucw1
#define WIDTH2  WW._eucw2
#define WIDTH3  WW._eucw3
#define MULTI_BYTE      WW._multibyte
#define GETONE(lc, p) \
        cw = ISASCII(lc = (unsigned char)*p++) ? 1 :     \
                (ISSET2(lc) ? WIDTH2 :                          \
                (ISSET3(lc) ? WIDTH3 : WIDTH1));                \
        if (--cw > --ccount) {                                  \
                cw -= ccount;                                   \
                while (ccount--)                                \
                        lc = (lc << 7) | ((*p++) & 0177);       \
                        if (p >= &buf[fw_lBufsiz + BUFSIZ]) {   \
                        if (nlp == buf) {                       \
                                /* Increase the buffer size */      \
                                fw_lBufsiz += BUFSIZ;                   \
                                if ((buf = realloc(buf,                 \
                                        fw_lBufsiz + BUFSIZ)) == NULL) { \
                                        exit(2); /* out of memory */    \
                                }                                       \
                                nlp = buf;                              \
                                p = &buf[fw_lBufsiz];                   \
                        } else {                                        \
                                /* shift the buffer contents down */ \
                                (void) memmove(buf, nlp,                \
                                        &buf[fw_lBufsiz + BUFSIZ] - nlp);\
                                p -= nlp - buf;                         \
                                nlp = buf;                              \
                        }                                               \
                }                                                       \
                if (p > &buf[fw_lBufsiz]) {                             \
                        if ((ccount = fread(p, sizeof (char),    \
                            &buf[fw_lBufsiz + BUFSIZ] - p, fptr))\
                                <= 0) break;                            \
                } else if ((ccount = fread(p,                           \
                        sizeof (char),  BUFSIZ, fptr)) <= 0)    \
                        break;                                          \
                blkno += (long long)ccount;                             \
        }                                                       \
        ccount -= cw;                                           \
        while (cw--)                                            \
                lc = (lc << 7) | ((*p++) & 0177)

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

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

#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;
int     retcode = 0;
int     nfile;
long long blkno;
int     nsucc;
long long tln;
FILE    *wordf;
char    *argptr;
off_t input_size = 0;

void    execute(char *);
void    cgotofn(void);
void    overflo(void);
void    cfail(void);

static long fw_lBufsiz = 0;

int
main(int argc, char **argv)
{
        int c;
        int errflg = 0;
        struct stat file_stat;

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

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

                case 'q':
                case 's': /* Solaris: legacy option */
                        qflag++;
                case 's':
                        sflag++;
                        continue;
                case 'H':
                        Hflag++;
                        hflag = 0;
                        continue;
                case 'h':
                        hflag++;
                        Hflag = 0;
                        continue;
                case 'b':
                        bflag++;
                        continue;

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

                case 'c':
                        cflag++;
                        continue;

                case 'e':
```



```
                        eflag++;
                        argptr = optarg;
                        input_size = strlen(argptr);
                        continue;

                case 'f':
                        fflag++;
                        wordf = fopen(optarg, "r");
                        if (wordf == NULL) {
                                (void) fprintf(stderr,
                                    gettext("fgrep: can't open %s\n"),
                                    optarg);
                                exit(2);
                        }

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

                        continue;

                case 'l':
                        lflag++;
                        continue;

                case 'n':
                        nflag++;
                        continue;

                case 'v':
                        vflag++;
                        continue;

                case 'x':
                        xflag++;
                        continue;

                case '?':
                        errflg++;
        }

        argc -= optind;
        if (errflg || ((argc <= 0) && !fflag && !eflag)) {
                (void) printf(gettext("usage: fgrep [ -bcHhilnqsvx ] "
                (void) printf(gettext("usage: fgrep [ -bchilnsvx ] "
                    "[ -e exp ] [ -f file ] [ strings ] [ file ] ...\n"));
                exit(2);
        }
        if (!eflag && !fflag) {
                argptr = argv[optind];
                input_size = strlen(argptr);
                optind++;
                argc--;
        }

/*
 * Normally we need one struct words for each letter in the pattern
 * plus one terminating struct words with outp = 1, but when -x option
 * is specified we require one more struct words for '\n' character so we
 * 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         }

257         input_size++;

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         }

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++;

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                 }

291         if (w != NULL) {
292                 free(w);
293         }

295         return (retcode != 0 ? retcode : nsucc == 0);
296 }

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;

310         if (buf == NULL) {
311                 fw_lBufsiz = BUFSIZ;
312                 if ((buf = malloc(fw_lBufsiz + BUFSIZ)) == NULL) {
313                         exit(2); /* out of memory */
314                 }
315         }
```



```
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 {
316                 file = "<stdin>";
325                 fptr = stdin;
326                 file = STDIN_FILENAME;
327         }
328         ccount = 0;
329         failed = 0;
330         lnum = 1;
331         tln = 0;
332         blkno = 0;
333         p = buf;
334         nlp = p;
335         c = w;
336         for (;;) {
337                 if (c == 0)
338                         break;
339                 if (ccount <= 0) {
340                         if (p >= &buf[fw_lBufsiz + BUFSIZ]) {
341                                 if (nlp == buf) {
342                                         /* increase the buffer size */
343                                         fw_lBufsiz += BUFSIZ;
344                                         if ((buf = realloc(buf,
345                                             fw_lBufsiz + BUFSIZ)) == NULL) {
346                                                 exit(2); /* out of memory */
347                                         }
348                                         nlp = buf;
349                                         p = &buf[fw_lBufsiz];
350                                 } else {
351                                         /* shift the buffer down */
352                                         (void) memmove(buf, nlp,
353                                             &buf[fw_lBufsiz + BUFSIZ]
354                                             - nlp);
355                                         p -= nlp - buf;
356                                         nlp = buf;
357                                 }

359                         }
360                         if (p > &buf[fw_lBufsiz]) {
361                                 if ((ccount = fread(p, sizeof (char),
362                                     &buf[fw_lBufsiz + BUFSIZ] - p, fptr))
363                                     <= 0)
364                                         break;
365                         } else if ((ccount = fread(p, sizeof (char),
366                             BUFSIZ, fptr)) <= 0)
367                                 break;
368                         blkno += (long long)ccount;
369                 }
370                 GETONE(lc, p);
371 nstate:
372                 if (same(c->inp, lc)) {
373                         c = c->nst;
374                 } else if (c->link != 0) {
375                         c = c->link;
376                         goto nstate;
377                 } else {
378                         c = c->fail;
379                         failed = 1;
380                         if (c == 0) {
```

```
                                c = w;
istate:
                                if (same(c->inp, lc)) {
                                        c = c->nst;
                                } else if (c->link != 0) {
                                        c = c->link;
                                        goto istate;
                                }
                        } else
                                goto nstate;
                }

                if (c == 0)
                        break;

                if (c->out) {
                        while (lc != '\n') {
                                if (ccount <= 0) {
if (p == &buf[fw_lBufsiz + BUFSIZ]) {
        if (nlp == buf) {
                /* increase 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 buffer down */
                (void) memmove(buf, nlp, &buf[fw_lBufsiz + BUFSIZ] - nlp);
                p -= nlp - buf;
                nlp = buf;
        }
}
```

_____unchanged_portion_omitted_

```
                        if ((vflag && (failed == 0 || xflag == 0)) ||
                                (vflag == 0 && xflag && failed))
                                goto nomatch;
succeed:
                        nsucc = 1;
                        if (lflag || qflag) {
                                if (!qflag)
                        if (cflag)
                                tln++;
                        else if (lflag && !sflag) {
                                (void) printf("%s\n", file);
                        (void) fclose(fptr);
                        return;
                        }
                        if (cflag) {
                                tln++;
                        } else {
                                if (Hflag || (nfile > 1 && !hflag))
                        } else if (!sflag) {
                                if (nfile > 1 && !hflag)
                                        (void) printf("%s:", file);
                                if (bflag)
                                        (void) printf("%lld:",
                                                (blkno - (long long)(ccount-1))
                                                / BUFSIZ);
                                if (nflag)
                                        (void) printf("%lld:", lnum);
                                if (p <= nlp) {
                                        while (nlp < &buf[fw_lBufsiz + BUFSIZ])
                                                (void) putchar(*nlp++);
                                        nlp = buf;
```



```
                                }
                                while (nlp < p)
                                        (void) putchar(*nlp++);
                        }
nomatch:
                        lnum++;
                        nlp = p;
                        c = w;
                        failed = 0;
                        continue;
                }
                if (lc == '\n')
                        if (vflag)
                                goto succeed;
                        else {
                                lnum++;
                                nlp = p;
                                c = w;
                                failed = 0;
                        }
        }
        (void) fclose(fptr);
        if (cflag && !qflag) {
                if (Hflag || (nfile > 1 && !hflag))
        if (cflag) {
                if ((nfile > 1) && !hflag)
                        (void) printf("%s:", file);
                (void) printf("%lld\n", tln);
        }
}
```

_____unchanged_portion_omitted_

```
**********************************************************
   10567 Fri Sep 13 10:33:21 2013
new/usr/src/cmd/grep/grep.c
3737 grep does not support -H option
3759 egrep(1) and fgrep(1) -s flag does not hide -c output
Reviewed by: Albert Lee <trisk@nexenta.com>
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
**********************************************************
```

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

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

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

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

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

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

#include <sys/types.h>

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



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

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

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

#define errmsg(msg, arg)        (void) fprintf(stderr, gettext(msg), arg)
#define BLKSIZE 512
#define GBUFSIZ 8192
#define MAX_DEPTH       1000

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

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

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

```
125         extern int      optind;

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

133         while ((c = getopt(argc, argv, "hHqblcnRrsviyw")) != -1)
126         while ((c = getopt(argc, argv, "hqblcnRrsviyw")) != -1)
134                 switch (c) {
135                 /* based on options order h or H is set as in GNU grep */
136                 case 'h':
137                         hflag++;
138                         Hflag = 0; /* h excludes H */
139                         break;
140                 case 'H':
141                         if (!lflag) /* H is excluded by l */
142                                 Hflag++;
143                         hflag = 0; /* H excludes h */
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                         break;
173                 case 'y':
174                 case 'i':
175                         iflag++;
176                         break;
177                 case 'w':
178                         wflag++;
179                         break;
180                 case '?':
181                         errflg++;
182                 }

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



```
189         }

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

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

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

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

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

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

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

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

229         return (nsucc == 2 ? 2 : (nsucc == 0 ? 1 : 0));
230 }
_____unchanged_portion_omitted_

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

305         tln = 0;

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

317         if (file == NULL) {
```

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

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

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

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

347                         /*
348                          * shift unused data to the beginning of the buffer
349                          */
350                         if (ptr > prntbuf) {
351                                 (void) memmove(prntbuf, ptr, offset);
352                                 ptr = prntbuf;
353                         }

355                         /*
356                          * re-allocate a larger buffer if this one is full
357                          */
358                         if (offset + GBUFSIZ > fw_lPrntBufLen) {
359                                 /*
360                                  * allocate a new buffer and preserve the
361                                  * contents...
362                                  */
363                                 fw_lPrntBufLen += GBUFSIZ;
364                                 if ((prntbuf = realloc(prntbuf,
365                                     fw_lPrntBufLen)) == NULL)
366                                         exit(2);

368                                 /*
369                                  * set up a bigger linebuffer (this is only used
370                                  * for case insensitive operations). Contents do
371                                  * not have to be preserved.
372                                  */
373                                 free(linebuf);
374                                 if ((linebuf = malloc(fw_lPrntBufLen)) == NULL)
375                                         exit(2);

377                                 ptr = prntbuf;
378                         }

380                         p = prntbuf + offset;
```



```
381                         if ((count = read(temp, p, GBUFSIZ)) > 0)
382                                 continue;

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

388                         /* last line of file has no newline */
389                         ptrend = ptr + offset;
390                         nlflag = 0;
391                 } else {
392                         next_ptr = ptrend + 1;
393                         next_count = offset + count - (next_ptr - ptr);
394                         nlflag = 1;
395                 }
396                 lnum++;
397                 *ptrend = '\0';

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

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

419                 if (!nlflag)
420                         break;

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

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

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

426         if (f == NULL)
427                 f = "<stdin>";
```

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

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

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

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

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

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

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

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

_____***unchanged_portion_omitted_***

```
**********************************************************
   28372 Fri Sep 13 10:33:21 2013
new/usr/src/cmd/grep_xpg4/grep.c
3737 grep does not support -H option
3759 egrep(1) and fgrep(1) -s flag does not hide -c output
Reviewed by: Albert Lee <trisk@nexenta.com>
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
**********************************************************
```

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

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

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

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

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

#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)")

#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 */
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
 */
```

```
125 int
126 main(int argc, char **argv)
127 {
128         char    *ap;
129         int     c;
130         int     fflag = 0;
131         int     i, n_pattern = 0, n_file = 0;
132         char    **pattern_list = NULL;
133         char    **file_list = NULL;

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

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

152         ap = cmdname;
153         /*
154          * Detect egrep/fgrep via command name, map to -E and -F options.
155          */
156         if (*ap == 'e' || *ap == 'E') {
157                 regflags |= REG_EXTENDED;
158                 egrep++;
159         } else {
160                 if (*ap == 'f' || *ap == 'F') {
161                         fgrep++;
162                 }
163         }

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

171                 case 'c':       /* POSIX: write count */
172                         cflag++;
173                         break;

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

180                 case 'l':       /* POSIX: Write filenames only */
181                         lflag++;
182                         break;

184                 case 'n':       /* POSIX: Write line numbers */
185                         nflag++;
186                         break;

188                 case 'r':       /* Solaris: search recursively */
189                         rflag++;
```



```
190                         break;

192                 case 'b':       /* Solaris: Write file block numbers */
193                         bflag++;
194                         break;

196                 case 's':       /* POSIX: No error msgs for files */
197                         sflag++;
198                         break;

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

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

227                 /* based on options order h or H is set as in GNU grep */
228                 case 'h':       /* Solaris: supress printing of file name */
229                         hflag = 1;
230                         Hflag = 0;
231                         break;
232                 /* Solaris: precede every matching with file name */
233                 case 'H':
234                         Hflag = 1;
235                         hflag = 0;
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          * -c, -l and -q flags are mutually exclusive
319          * We have -c override -l like in Solaris.
320          * -q overrides -l & -c programmatically in grep() function.
321          */
```



```
322         if (cflag && lflag)
323                 lflag = 0;

325         argv += optind - 1;
326         argc -= optind - 1;

328         /*
329          * Now handling -e and -f option
330          */
331         if (pattern_list) {
332                 for (i = 0; i < n_pattern; i++) {
333                         addpattern(pattern_list[i]);
334                 }
335                 free(pattern_list);
336         }
337         if (file_list) {
338                 for (i = 0; i < n_file; i++) {
339                         addfile(file_list[i]);
340                 }
341                 free(file_list);
342         }

344         /*
345          * No -e or -f?  Make sure there is one more arg, use it as the pattern.
346          */
347         if (patterns == NULL && !fflag) {
348                 if (argc < 2)
349                         usage();
350                 addpattern(argv[1]);
351                 argc--;
352                 argv++;
353         }

355         /*
356          * If -x flag is not specified or -i flag is specified
357          * with fgrep in a multibyte locale, need to use
358          * the wide character APIs.  Otherwise, byte-oriented
359          * process will be done.
360          */
361         use_wchar = Fflag && mblocale && (!xflag || iflag);

363         /*
364          * Compile Patterns and also decide if BMG can be used
365          */
366         fixpatterns();

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

382         (void) fflush(stdout);

384         if (errors)
385                 return (2);
```

```
 386         return (matched ? 0 : 1);
 387 }
_____unchanged_portion_omitted_

 788 /*
 789  * Do grep on a single file.
 790  * Return true in any lines matched.
 791  *
 792  * We have two strategies:
 793  * The fast one is used when we have a single pattern with
 794  * a string known to occur in the pattern. We can then
 795  * do a BMG match on the whole buffer.
 796  * This is an order of magnitude faster.
 797  * Otherwise we split the buffer into lines,
 798  * and check for a match on each line.
 799  */
 800 static int
 801 grep(int fd, const char *fn)
 802 {
 803         PATTERN *pp;
 804         off_t   data_len;       /* length of the data chunk */
 805         off_t   line_len;       /* length of the current line */
 806         off_t   line_offset;    /* current line's offset from the beginning */
 807         long long       lineno;
 808         long long       matches = 0;    /* Number of matching lines */
 809         int     newlinep;       /* 0 if the last line of file has no newline */
 810         char    *ptr, *ptrend;


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

 816         pp = patterns;

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

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

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

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

 849                 if (data_len == 0) {
```



```
 850                         /*
 851                          * If no data in the buffer, reset ptr
 852                          */
 853                         ptr = prntbuf;
 854                 }
 855                 if (ptr == prntbuf) {
 856                         /*
 857                          * The current data chunk starts from prntbuf.
 858                          * This means either the buffer has no data
 859                          * or the buffer has no newline.
 860                          * So, read more data from input.
 861                          */
 862                         count = read(fd, ptr + data_len, prntbuflen - data_len);
 863                         if (count < 0) {
 864                                 /* read error */
 865                                 if (cflag) {
 866                                         if (outfn && !rflag) {
 867                                                 (void) fprintf(stdout,
 868                                                     "%s:", fn);
 869                                         }
 870                                         if (!qflag && !rflag) {
 871                                                 (void) fprintf(stdout, "%lld\n",
 872                                                     matches);
 873                                         }
 874                                 }
 875                                 return (0);
 876                         } else if (count == 0) {
 877                                 /* no new data */
 878                                 if (data_len == 0) {
 879                                         /* end of file already reached */
 880                                         break;
 881                                 }
 882                                 /* last line of file has no newline */
 883                                 ptrend = ptr + data_len;
 884                                 newlinep = 0;
 885                                 goto L_start_process;
 886                         }
 887                         offset = data_len;
 888                         data_len += count;
 889                 }

 891                 /*
 892                  * Look for newline in the chunk
 893                  * between ptr + offset and ptr + data_len - offset.
 894                  */
 895                 ptrend = find_nl(ptr + offset, data_len - offset);
 896                 if (ptrend == NULL) {
 897                         /* no newline found in this chunk */
 898                         if (ptr > prntbuf) {
 899                                 /*
 900                                  * Move remaining data to the beginning
 901                                  * of the buffer.
 902                                  * Remaining data lie from ptr for
 903                                  * data_len bytes.
 904                                  */
 905                                 (void) memmove(prntbuf, ptr, data_len);
 906                         }
 907                         if (data_len == prntbuflen) {
 908                                 /*
 909                                  * No enough room in the buffer
 910                                  */
 911                                 prntbuflen += BUFSIZE;
 912                                 prntbuf = realloc(prntbuf, prntbuflen + 1);
 913                                 if (prntbuf == NULL) {
 914                                         (void) fprintf(stderr,
 915                                             gettext("%s: out of memory\n"),
```

```
916                                         cmdname);
917                                 exit(2);
918                         }
919                 }
920                 ptr = prntbuf;
921                 /* read the next input */
922                 continue;
923         }
924 L_start_process:

926         /*
927          * Beginning of the chunk:      ptr
928          * End of the chunk:            ptr + data_len
929          * Beginning of the line:       ptr
930          * End of the line:             ptrend
931          */

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

950                         /*
951                          * When this chunk does not contain newline,
952                          * ptrend becomes NULL, which should happen
953                          * when the last line of file does not end
954                          * with a newline.  At such a point,
955                          * newlinep should have been set to 0.
956                          * Therefore, just after jumping to
957                          * L_skip_line, the main for-loop quits,
958                          * and the line_len value won't be
959                          * used.
960                          */
961                         line_len = ptrend - ptr;
962                         goto L_skip_line;
963                 }
964                 if (bline > ptrend) {
965                         /*
966                          * Pattern found not in the first line
967                          * of this chunk.
968                          * Discard the first line.
969                          */
970                         line_len = ptrend - ptr;
971                         goto L_skip_line;
972                 }
973                 /*
974                  * Pattern found in the first line of this chunk.
975                  * Using this result.
976                  */
977                 *ptrend = '\0';
978                 line_len = ptrend - ptr;

980                 /*
981                  * before jumping to L_next_line,
```



```
982                  * need to handle xflag if specified
983                  */
984                 if (xflag && (line_len != bmglen ||
985                     strcmp(bmgpat, ptr) != 0)) {
986                         /* didn't match */
987                         pp = NULL;
988                 } else {
989                         pp = patterns; /* to make it happen */
990                 }
991                 goto L_next_line;
992         }
993         lineno++;
994         /*
995          * Line starts from ptr and ends at ptrend.
996          * line_len will be the length of the line.
997          */
998         *ptrend = '\0';
999         line_len = ptrend - ptr;

1001        /*
1002         * From now, the process will be performed based
1003         * on the line from ptr to ptrend.
1004         */
1005        if (use_wchar) {
1006                size_t  len;

1008                if (line_len >= outbuflen) {
1009                        outbuflen = line_len + 1;
1010                        outline = realloc(outline,
1011                            sizeof (wchar_t) * outbuflen);
1012                        if (outline == NULL) {
1013                                (void) fprintf(stderr,
1014                                    gettext("%s: out of memory\n"),
1015                                    cmdname);
1016                                exit(2);
1017                        }
1018                }

1020                len = mbstowcs(outline, ptr, line_len);
1021                if (len == (size_t)-1) {
1022                        (void) fprintf(stderr, gettext(
1023        "%s: input file \"%s\": line %lld: invalid multibyte character\n"),
1024                            cmdname, fn, lineno);
1025                        /* never match a line with invalid sequence */
1026                        goto L_skip_line;
1027                }
1028                outline[len] = L'\0';

1030                if (iflag) {
1031                        wchar_t *cp;
1032                        for (cp = outline; *cp != '\0'; cp++) {
1033                                *cp = towlower((wint_t)*cp);
1034                        }
1035                }

1037                if (xflag) {
1038                        for (pp = patterns; pp; pp = pp->next) {
1039                                if (outline[0] == pp->wpattern[0] &&
1040                                    wcscmp(outline,
1041                                    pp->wpattern) == 0) {
1042                                        /* matched */
1043                                        break;
1044                                }
1045                        }
1046                } else {
1047                        for (pp = patterns; pp; pp = pp->next) {
```

```
1048                                    if (wcswcs(outline, pp->wpattern)
1049                                        != NULL) {
1050                                            /* matched */
1051                                            break;
1052                                    }
1053                            }
1054                    }
1055            } else if (Fflag) {
1056                    /* fgrep in byte-oriented handling */
1057                    char    *fptr;
1058                    if (iflag) {
1059                            fptr = istrdup(ptr);
1060                    } else {
1061                            fptr = ptr;
1062                    }
1063                    if (xflag) {
1064                            /* fgrep -x */
1065                            for (pp = patterns; pp; pp = pp->next) {
1066                                    if (fptr[0] == pp->pattern[0] &&
1067                                        strcmp(fptr, pp->pattern) == 0) {
1068                                            /* matched */
1069                                            break;
1070                                    }
1071                            }
1072                    } else {
1073                            for (pp = patterns; pp; pp = pp->next) {
1074                                    if (strstr(fptr, pp->pattern) != NULL) {
1075                                            /* matched */
1076                                            break;
1077                                    }
1078                            }
1079                    }
1080            } else {
1081                    /* grep or egrep */
1082                    for (pp = patterns; pp; pp = pp->next) {
1083                            int     rv;

1085                            rv = regexec(&pp->re, ptr, 0, NULL, 0);
1086                            if (rv == REG_OK) {
1087                                    /* matched */
1088                                    break;
1089                            }

1091                            switch (rv) {
1092                            case REG_NOMATCH:
1093                                    break;
1094                            case REG_ECHAR:
1095                                    (void) fprintf(stderr, gettext(
1096            "%s: input file \"%s\": line %lld: invalid multibyte character\n"),
1097                                        cmdname, fn, lineno);
1098                                    break;
1099                            default:
1100                                    (void) regerror(rv, &pp->re, errstr,
1101                                        sizeof (errstr));
1102                                    (void) fprintf(stderr, gettext(
1103            "%s: input file \"%s\": line %lld: %s\n"),
1104                                        cmdname, fn, lineno, errstr);
1105                                    exit(2);
1106                            }
1107                    }
1108            }

1110 L_next_line:
1111                 /*
1112                  * Here, if pp points to non-NULL, something has been matched
1113                  * to the pattern.
```



```
1114                  */
1115                 if (nvflag == (pp != NULL)) {
1116                         matches++;
1117                         /*
1118                          * Handle q, l, and c flags.
1119                          */
1120                         if (qflag) {
1121                                 /* no need to continue */
1122                                 /*
1123                                  * End of this line is ptrend.
1124                                  * We have read up to ptr + data_len.
1125                                  */
1126                                 off_t   pos;
1127                                 pos = ptr + data_len - (ptrend + 1);
1128                                 (void) lseek(fd, -pos, SEEK_CUR);
1129                                 exit(0);
1130                         }
1131                         if (lflag) {
1132                                 (void) printf("%s\n", fn);
1133                                 break;
1134                         }
1135                         if (!cflag) {
1136                                 if (Hflag || outfn) {
1115                                 if (outfn) {
1137                                         (void) printf("%s:", fn);
1138                                 }
1139                                 if (bflag) {
1140                                         (void) printf("%lld:", (offset_t)
1141                                             (line_offset / BSIZE));
1142                                 }
1143                                 if (nflag) {
1144                                         (void) printf("%lld:", lineno);
1145                                 }
1146                                 *ptrend = '\n';
1147                                 (void) fwrite(ptr, 1, line_len + 1, stdout);
1148                         }
1149                         if (ferror(stdout)) {
1150                                 return (0);
1151                         }
1152                 }
1153 L_skip_line:
1154                 if (!newlinep)
1155                         break;

1157                 data_len -= line_len + 1;
1158                 line_offset += line_len + 1;
1159                 ptr = ptrend + 1;
1160         }

1162         if (cflag) {
1163                 if (Hflag || outfn) {
1142                 if (outfn) {
1164                         (void) printf("%s:", fn);
1165                 }
1166                 if (!qflag) {
1167                         (void) printf("%lld\n", matches);
1168                 }
1169         }
1170         return (matches != 0);
1171 }

1173 /*
1174  * usage message for grep
1175  */
1176 static void
1177 usage(void)
```

```
1178 {
1179         if (egrep || fgrep) {
1180                 (void) fprintf(stderr, gettext("Usage:\t%s"), cmdname);
1181                 (void) fprintf(stderr,
1182                     gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvx] "
1161                     gettext(" [-c|-l|-q] [-r|-R] [-bhinsvx] "
1183                     "pattern_list [file ...]\n"));

1185                 (void) fprintf(stderr, "\t%s", cmdname);
1186                 (void) fprintf(stderr,
1187                     gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvx] "
1166                     gettext(" [-c|-l|-q] [-r|-R] [-bhinsvx] "
1188                     "[-e pattern_list]... "
1189                     "[-f pattern_file]... [file...]\n"));
1190         } else {
1191                 (void) fprintf(stderr, gettext("Usage:\t%s"), cmdname);
1192                 (void) fprintf(stderr,
1193                     gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvwx] "
1172                     gettext(" [-c|-l|-q] [-r|-R] [-bhinsvwx] "
1194                     "pattern_list [file ...]\n"));

1196                 (void) fprintf(stderr, "\t%s", cmdname);
1197                 (void) fprintf(stderr,
1198                     gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvwx] "
1177                     gettext(" [-c|-l|-q] [-r|-R] [-bhinsvwx] "
1199                     "[-e pattern_list]... "
1200                     "[-f pattern_file]... [file...]\n"));

1202                 (void) fprintf(stderr, "\t%s", cmdname);
1203                 (void) fprintf(stderr,
1204                     gettext(" -E [-c|-l|-q] [-r|-R] [-bhHinsvx] "
1183                     gettext(" -E [-c|-l|-q] [-r|-R] [-bhinsvx] "
1205                     "pattern_list [file ...]\n"));

1207                 (void) fprintf(stderr, "\t%s", cmdname);
1208                 (void) fprintf(stderr,
1209                     gettext(" -E [-c|-l|-q] [-r|-R] [-bhHinsvx] "
1188                     gettext(" -E [-c|-l|-q] [-r|-R] [-bhinsvx] "
1210                     "[-e pattern_list]... "
1211                     "[-f pattern_file]... [file...]\n"));

1213                 (void) fprintf(stderr, "\t%s", cmdname);
1214                 (void) fprintf(stderr,
1215                     gettext(" -F [-c|-l|-q] [-r|-R] [-bhHinsvx] "
1194                     gettext(" -F [-c|-l|-q] [-r|-R] [-bhinsvx] "
1216                     "pattern_list [file ...]\n"));

1218                 (void) fprintf(stderr, "\t%s", cmdname);
1219                 (void) fprintf(stderr,
1220                     gettext(" -F [-c|-l|-q] [-bhHinsvx] [-e pattern_list]... "
1199                     gettext(" -F [-c|-l|-q] [-bhinsvx] [-e pattern_list]... "
1221                     "[-f pattern_file]... [file...]\n"));
1222         }
1223         exit(2);
1224         /* NOTREACHED */
1225 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    9118 Fri Sep 13 10:33:21 2013
new/usr/src/man/man1/egrep.1
3737 grep does not support -H option
3759 egrep(1) and fgrep(1) -s flag does not hide -c output
Reviewed by: Albert Lee <trisk@nexenta.com>
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
**********************************************************
'\" 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
\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
\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 Fri Sep 13 10:33:21 2013
new/usr/src/man/man1/fgrep.1
3737 grep does not support -H option
3759 egrep(1) and fgrep(1) -s flag does not hide -c output
Reviewed by: Albert Lee <trisk@nexenta.com>
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
**********************************************************
'\" 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
\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
```

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

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

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

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

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

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



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

```
**********************************************************
   14031 Fri Sep 13 10:33:21 2013
new/usr/src/man/man1/grep.1
3737 grep does not support -H option
3759 egrep(1) and fgrep(1) -s flag does not hide -c output
Reviewed by: Albert Lee <trisk@nexenta.com>
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
**********************************************************
'\" 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
\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
```

```
516 _
517 Standard        See \fBstandards\fR(5).
518 .TE

520 .SH SEE ALSO
521 .sp
522 .LP
523 \fBegrep\fR(1), \fBfgrep\fR(1), \fBsed\fR(1), \fBsh\fR(1), \fBattributes\fR(5),
524 \fBenviron\fR(5), \fBlargefile\fR(5), \fBregex\fR(5), \fBregexp\fR(5),
525 \fBstandards\fR(5)
526 .SH NOTES
527 .SS "/usr/bin/grep"
528 .sp
529 .LP
530 Lines are limited only by the size of the available virtual memory. If there is
531 a line with embedded nulls, \fBgrep\fR only matches up to the first null. If
532 the line matches, the entire line is printed.
533 .SS "/usr/xpg4/bin/grep"
534 .sp
535 .LP
536 The results are unspecified if input files contain lines longer than
537 \fBLINE_MAX\fR bytes or contain binary data. \fBLINE_MAX\fR is defined in
538 \fB/usr/include/limits.h\fR.
```