

```

*****
26107 Thu Sep 12 13:22:55 2013
new/usr/src/cmd/egrep/egrep.y
3737 grep does not support -H option
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
*****
1  %{
2  /*
3   * CDDL HEADER START
4   *
5   * The contents of this file are subject to the terms of the
6   * Common Development and Distribution License, Version 1.0 only
7   * (the "License"). You may not use this file except in compliance
8   * with the License.
9   *
10  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
11  * or http://www.opensolaris.org/os/licensing.
12  * See the License for the specific language governing permissions
13  * and limitations under the License.
14  *
15  * When distributing Covered Code, include this CDDL HEADER in each
16  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
17  * If applicable, add the following below this CDDL HEADER, with the
18  * fields enclosed by brackets "[]" replaced with your own identifying
19  * information: Portions Copyright [yyyy] [name of copyright owner]
20  *
21  * CDDL HEADER END
22  */
23  %}
24  /*
25   * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
26   * Use is subject to license terms.
27   */

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

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

35  /*
36   * Copyright 2013 Damian Bogel. All rights reserved.
37   */
35  %{
36  #pragma ident    "%Z%M% %I%    %E% SMI"
37  %}

39  /*
40   * egrep -- print lines containing (or not containing) a regular expression
41   *
42   *      status returns:
43   *          0 - ok, and some matches
44   *          1 - ok, but no matches
45   *          2 - some error; matches irrelevant
46   */
47  %token CHAR MCHAR DOT MDOT CCL NCCL MCCL NMCCCL OR CAT STAR PLUS QUEST
48  %left OR
49  %left CHAR MCHAR DOT CCL NCCL MCCL NMCCCL '('
50  %left CAT
51  %left STAR PLUS QUEST

53  %{
54  #include <stdio.h>
55  #include <ctype.h>
56  #include <memory.h>
57  #include <wchar.h>

```

```

58  #include <wctype.h>
59  #include <wider.h>
60  #include <stdlib.h>
61  #include <limits.h>
62  #include <locale.h>

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

66  #define BLKSIZE 512      /* size of reported disk blocks */
67  #define EBUFSIZ 8192
68  #define MAXLIN 350
69  #define NCHARS 256
70  #define MAXPOS 4000
71  #define NSTATES 64
72  #define FINAL -1
73  #define RIGHT '\n'     /* serves as record separator and as $ */
74  #define LEFT '\n'     /* beginning of line */
75  int gotofn[NSTATES][NCHARS];
76  int state[NSTATES];
77  int out[NSTATES];
78  int line = 1;
79  int *name;
80  int *left;
81  int *right;
82  int *parent;
83  int *foll;
84  int *positions;
85  char *chars;
86  wchar_t *lower;
87  wchar_t *upper;
88  int maxlin, maxclin, maxwclin, maxpos;
89  int nxtpos = 0;
90  int inxtpos;
91  int nxtchar = 0;
92  int *tmpstat;
93  int *initstat;
94  int istat;
95  int nstate = 1;
96  int xstate;
97  int count;
98  int icount;
99  char *input;

102 wchar_t llylval;
103 wchar_t nextch();
104 wchar_t maxmin();
105 int compare();
106 void overflo();

108 char reinit = 0;

110 long long lnum;
111 int    bflag;
112 int    cflag;
113 int    eflag;
114 int    fflag;
115 int    Hflag;
116 int    hflag;
117 int    iflag;
118 int    lflag;
119 int    nflag;
120 int    qflag;
121 int    sflag;
122 int    vflag;
122 int    nfile;

```



```

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++;
720         case 's':
721             sflag++;
722             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         overflow();
744     if ((left = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
745         overflow();
746     if ((right = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
747         overflow();
748     if ((parent = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
749         overflow();
750     if ((foll = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
751         overflow();
752     if ((tmpstat = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)
753         overflow();
754     if ((initstat = (int *)malloc(MAXLIN*sizeof(int))) == (int *)0)

```

```

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

771     cfollow(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 nlfag;

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 {
814         file = "<stdin>";
815         f = stdin;
816         file = STDIN_FILENAME;
817     }
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                         NULL) 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             nflag = 0;
866         } else
867             nflag = 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 = nextst(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) {
918     if(succ) {
919         nsucc = 1;
920         if (lflag || qflag) {
921             if (!qflag)
922                 (void) printf("%s\n", file);
923             if (cflag) tln++;
924             else if (sflag) /* ugh */
925                 ;
926             else if (lflag) {
927                 printf("%s\n", file);
928                 fclose(f);
929                 return;
930             }
931         }
932         if (cflag) {
933             tln++;
934         }
935         } else {
936             if (Hflag || (nfile > 1 && !hflag))
937                 printf("%s:", file);
938             else {
939                 if (nfile > 1 && !hflag)
940                     printf(gettext("%s:"), file);
941                 if (bflag) {
942                     nchars = blkno - (buf + count - ptrend)
943                     if(nlflag)
944                         nchars++;
945                     printf("%lld:", nchars/BLKSIZE);
946                 }
947                 if (nlflag)
948                     printf("%lld:", lnum);
949                 if(nlflag)
950                     nchars = ptrend - ptr + 1;
951             }
952             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);
962     if (cflag) {
963         if (nfile > 1 && !hflag)
964             printf(gettext("%s:"), file);
965         printf("%lld\n", tln);
966     }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }
1001 }
1002 }
1003 }
1004 }
1005 }
1006 }
1007 }
1008 }
1009 }
1010 }
1011 }
1012 }
1013 }
1014 }
1015 }
1016 }
1017 }
1018 }
1019 }
1020 }
1021 }
1022 }
1023 }
1024 }
1025 }
1026 }
1027 }
1028 }
1029 }
1030 }
1031 }
1032 }
1033 }
1034 }
1035 }
1036 }
1037 }
1038 }
1039 }
1040 }
1041 }
1042 }
1043 }
1044 }
1045 }
1046 }
1047 }
1048 }
1049 }
1050 }
1051 }
1052 }
1053 }
1054 }
1055 }
1056 }
1057 }
1058 }
1059 }
1060 }
1061 }
1062 }
1063 }
1064 }
1065 }
1066 }
1067 }
1068 }
1069 }
1070 }
1071 }
1072 }
1073 }
1074 }
1075 }
1076 }
1077 }
1078 }
1079 }
1080 }
1081 }
1082 }
1083 }
1084 }
1085 }
1086 }
1087 }
1088 }
1089 }
1090 }
1091 }
1092 }
1093 }
1094 }
1095 }
1096 }
1097 }
1098 }
1099 }
1100 }
1101 }
1102 }
1103 }
1104 }
1105 }
1106 }
1107 }
1108 }
1109 }
1110 }
1111 }
1112 }
1113 }
1114 }
1115 }
1116 }
1117 }
1118 }
1119 }
1120 }
1121 }
1122 }
1123 }
1124 }
1125 }
1126 }
1127 }
1128 }
1129 }
1130 }
1131 }
1132 }
1133 }
1134 }
1135 }
1136 }
1137 }
1138 }
1139 }
1140 }
1141 }
1142 }
1143 }
1144 }
1145 }
1146 }
1147 }
1148 }
1149 }
1150 }
1151 }
1152 }
1153 }
1154 }
1155 }
1156 }
1157 }
1158 }
1159 }
1160 }
1161 }
1162 }
1163 }
1164 }
1165 }
1166 }
1167 }
1168 }
1169 }
1170 }
1171 }
1172 }
1173 }
1174 }
1175 }
1176 }
1177 }
1178 }
1179 }
1180 }
1181 }
1182 }
1183 }
1184 }
1185 }
1186 }
1187 }
1188 }
1189 }
1190 }
1191 }
1192 }
1193 }
1194 }
1195 }
1196 }
1197 }
1198 }
1199 }
1200 }
```

\_\_\_\_\_unchanged\_portion\_omitted\_\_\_\_\_

```

*****
14491 Thu Sep 12 13:22:55 2013
new/usr/src/cmd/fgrep/fgrep.c
3737 grep does not support -H option
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License, Version 1.0 only
6  * (the "License").  You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
24 * Use is subject to license terms.
25 */

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

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

33 /*
34  * Copyright 2013 Damian Bogel. All rights reserved.
35  */
36 #pragma ident "%Z%M% %I% %E% SMI"

37 /*
38  * fgrep -- print all lines containing any of a set of keywords
39  *
40  *      status returns:
41  *          0 - ok, and some matches
42  *          1 - ok, but no matches
43  *          2 - some error
44  */

46 #include <stdio.h>
47 #include <ctype.h>
48 #include <sys/types.h>
49 #include <stdlib.h>
50 #include <string.h>
51 #include <locale.h>
52 #include <libintl.h>
53 #include <euc.h>
54 #include <sys/stat.h>
55 #include <fcntl.h>

57 #include <getwidth.h>

59 eucwidth_t WW;

```

```

60 #define WIDTH1  WW._eucw1
61 #define WIDTH2  WW._eucw2
62 #define WIDTH3  WW._eucw3
63 #define MULTI_BYTE  WW._multibyte
64 #define GETONE(lc, p) \
65     cw = ISASCII(lc = (unsigned char)*p++) ? 1 : \
66         (ISSET2(lc) ? WIDTH2 : \
67          (ISSET3(lc) ? WIDTH3 : WIDTH1));
68     if (--cw > --ccount) { \
69         cw -= ccount; \
70         while (ccount--) \
71             lc = (lc << 7) | ((*p++) & 0177); \
72         if (p >= &buf[fw_lBufsiz + BUFSIZ]) { \
73             if (nlp == buf) { \
74                 /* Increase the buffer size */ \
75                 fw_lBufsiz += BUFSIZ; \
76                 if ((buf = realloc(buf, \
77                  fw_lBufsiz + BUFSIZ)) == NULL) { \
78                     exit(2); /* out of memory */ \
79                 } \
80                 nlp = buf; \
81                 p = &buf[fw_lBufsiz]; \
82             } else { \
83                 /* shift the buffer contents down */ \
84                 (void) memmove(buf, nlp, \
85                  &buf[fw_lBufsiz + BUFSIZ] - nlp); \
86                 p -= nlp - buf; \
87                 nlp = buf; \
88             } \
89         } \
90     if (p > &buf[fw_lBufsiz]) { \
91         if ((ccount = fread(p, sizeof (char), \
92          &buf[fw_lBufsiz + BUFSIZ] - p, fptr)) \
93             <= 0) break; \
94     } else if ((ccount = fread(p, \
95          sizeof (char), BUFSIZ, fptr)) <= 0) \
96         break; \
97     blkno += (long long)ccount; \
98     } \
99     ccount -= cw; \
100    while (cw--) \
101        lc = (lc << 7) | ((*p++) & 0177)

103 /*
104  * The same() macro and letter() function were inserted to allow for
105  * the -i option work for the multi-byte environment.
106  */
107 wchar_t letter();
108 #define same(a, b) \
109     (a == b || iflag && (!MULTI_BYTE || ISASCII(a)) && (a ^ b) == ' ' && \
110      letter(a) == letter(b))

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

114 #define QSIZE 400
115 struct words {
116     wchar_t inp;
117     char out;
118     struct words *nst;
119     struct words *link;
120     struct words *fail;
121 } *w = NULL, *smax, *q;

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

```

```

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

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

141 static long fw_lBufsiz = 0;

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

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

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

159             case 'q':
160             case 's': /* Solaris: legacy option */
161                 qflag++;
162             case 's':
163                 sflag++;
164                 continue;
165             case 'H':
166                 Hflag++;
167                 hflag = 0;
168                 continue;
169             case 'h':
170                 hflag++;
171                 Hflag = 0;
172                 continue;
173             case 'b':
174                 bflag++;
175                 continue;
176             case 'i':
177             case 'y':
178                 iflag++;
179                 continue;

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

184             case 'e':
185                 eflag++;
186                 argptr = optarg;

```

```

187         input_size = strlen(argptr);
188         continue;

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

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

209         continue;

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

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

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

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

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

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

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

```

```

253     if (xflag) {
254         input_size = input_size + (input_size/2) + 1;
255     }

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 {
325         file = "<stdin>";
326         fptr = stdin;
327         file = STDIN_FILENAME;
328     }
329     ccount = 0;
330     failed = 0;
331     lnum = 1;
332     tln = 0;
333     blkno = 0;
334     p = buf;
335     nlp = p;
336     c = w;
337     for (;;) {
338         if (c == 0)
339             break;
340         if (ccount <= 0) {
341             if (p >= &buf[fw_lBufsiz + BUFSIZ]) {
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             }
358         }
359         if (p > &buf[fw_lBufsiz]) {
360             if ((ccount = fread(p, sizeof (char),
361                 &buf[fw_lBufsiz + BUFSIZ] - p, fptr))
362                 <= 0)
363                 break;
364             } else if ((ccount = fread(p, sizeof (char),
365                 BUFSIZ, fptr)) <= 0)
366                 break;
367             blkno += (long long)ccount;
368         }
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) {
381             c = w;
382             istate:

```





```

*****
10567 Thu Sep 12 13:22:55 2013
new/usr/src/cmd/grep/grep.c
3737 grep does not support -H option
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License, Version 1.0 only
6  * (the "License").  You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
24 * Use is subject to license terms.
25 */

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

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

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

35 /*
36  * Copyright 2013 Damian Bogel. All rights reserved.
37  */

39 /*
40  * grep -- print lines matching (or not matching) a pattern
41  *
42  *      status returns:
43  *          0 - ok, and some matches
44  *          1 - ok, but no matches
45  *          2 - some error
46  */

48 #include <sys/types.h>

50 #include <ctype.h>
51 #include <fcntl.h>
52 #include <locale.h>
53 #include <memory.h>
54 #include <regex.h>
55 #include <stdio.h>
56 #include <stdlib.h>
57 #include <string.h>
58 #include <unistd.h>
59 #include <ftw.h>
60 #include <limits.h>

```

```

61 #include <sys/param.h>

63 static const char *errstr[] = {
64     "Range endpoint too large.",
65     "Bad number.",
66     "'\\digit' out of range.",
67     "No remembered search string.",
68     "\\( \\) imbalance.",
69     "Too many \\(.",
70     "More than 2 numbers given in \\{ \\}.",
71     "} expected after \\.",
72     "First number exceeds second in \\{ \\}.",
73     "[ ] imbalance.",
74     "Regular expression overflow.",
75     "Illegal byte sequence.",
76     "Unknown regexp error code!!",
77     NULL
78 };

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

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

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

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

120 int
121 main(int argc, char **argv)
122 {
123     int      c;
124     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)
134     while ((c = getopt(argc, argv, "hHqblcnRrsviyw")) != -1)
135     switch (c) {
136     /* based on options order h or H is set as in GNU grep */
137     case 'h':
138         hflag++;
139         Hflag = 0; /* h excludes H */
140         break;
141     case 'H':
142         if (!lflag) /* H is excluded by l */
143             Hflag++;
144         hflag = 0; /* H excludes h */
145         break;
146     case 'q': /* POSIX: quiet: status only */
147         qflag++;
148         break;
149     case 'v':
150         vflag++;
151         break;
152     case 'c':
153         cflag++;
154         break;
155     case 'n':
156         nflag++;
157         break;
158     case 'R':
159         Rflag++;
160         /* FALLTHROUGH */
161     case 'r':
162         rflag++;
163         break;
164     case 'b':
165         bflag++;
166         break;
167     case 's':
168         sflag++;
169         break;
170     case 'l':
171         lflag++;
172         Hflag = 0; /* l excludes H */
173         break;
174     case 'y':
175     case 'i':
176         iflag++;
177         break;
178     case 'w':
179         wflag++;
180         break;
181     case '?':
182         errflg++;
183     }

184     if (errflg || (optind >= argc)) {
185         errmsg("Usage: grep [-c|-l|-q] [-r|-R] -hHbnsviw "
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))
333             if (nfile > 1 && !hflag && file)
334                 (void) fprintf(stdout, "%s:", file);
335             if (!rflag)
336                 (void) fprintf(stdout, "%lld\n", tln);
337     }
338     return;

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))))
431         if (!hflag && file && (nfile > 1 ||
432                              (rflag && outfn)))
433             (void) fprintf(stdout, "%s:", file);
434 }

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

442     if (f == NULL)
443         f = "<stdin>";

444     if (qflag) {
445         /* 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 Thu Sep 12 13:22:55 2013
new/usr/src/cmd/grep_xpg4/grep.c
3737 grep does not support -H option
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License, Version 1.0 only
6  * (the "License").  You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
24 * Use is subject to license terms.
25 */

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

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

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

39 /*
40 * Copyright 2013 Damian Bogel. All rights reserved.
41 */

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

```

```

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

63 #define BSIZE 512 /* Size of block for -b */
64 #define BUFSIZE 8192 /* Input buffer size */
65 #define MAX_DEPTH 1000 /* how deep to recurse */

67 #define M_CSETSIZE 256 /* singlebyte chars */
68 static int bmglen; /* length of BMG pattern */
69 static char *bmgpat; /* BMG pattern */
70 static int bmgtab[M_CSETSIZE]; /* BMG delta table */

72 typedef struct _PATTERN {
73     char *pattern; /* original pattern */
74     wchar_t *wpattern; /* wide, lowercased pattern */
75     struct _PATTERN *next;
76     regex_t re; /* compiled pattern */
77 } PATTERN;

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

105 static int use_wchar, use_bmg, mblocale;

107 static size_t outbuflen, prntbuflen;
108 static char *prntbuf;
109 static wchar_t *outline;

111 static void addfile(const char *fn);
112 static void addpattern(char *s);
113 static void fixpatterns(void);
114 static void usage(void);
115 static int grep(int, const char *);
116 static void bmgcomp(char *, int);
117 static char *bmgexec(char *, char *);
118 static int recursive(const char *, const struct stat *, int, struct FTW *);
119 static void process_path(const char *);
120 static void process_file(const char *, int);

122 /*
123 * mainline for grep
124 */
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, "vwchHilnrbs:f:qxEFIR")) != EOF) {
166     while ((c = getopt(argc, argv, "vwchilnrbs:f:qxEFIR")) != EOF) {
167         switch (c) {
168             case 'v': /* POSIX: negate matches */
169                 nvflag = 0;
170                 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: suppress 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
295         Fflag++; /* Else fgrep */
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);
371         matched = grep(0, gettext("standard input"));
372     } else {
373         if (Hflag || (argc > 2 && hflag == 0))
374             if (argc > 2 && hflag == 0)
375                 outf = 1; /* Print filename on match line */
376         for (argv++; *argv != NULL; argv++) {
377             process_path(*argv);
378         }
379     }
380     /*
381      * Return() here is used instead of exit
382      */
383     (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:
925
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  */
932
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);
949
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;
979
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;
1000
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;
1007
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         }
1019
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';
1029
1030         if (iflag) {
1031             wchar_t *cp;
1032             for (cp = outline; *cp != '\0'; cp++) {
1033                 *cp = towlower((wint_t)*cp);
1034             }
1035         }
1036
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     } else if (Fflag) {
1055     /* fgrep in byte-oriented handling */
1056     char *fptr;
1057     if (iflag) {
1058         fptr = istrdup(ptr);
1059     } else {
1060         fptr = ptr;
1061     }
1062     if (xflag) {
1063         /* fgrep -x */
1064         for (pp = patterns; pp; pp = pp->next) {
1065             if (fptr[0] == pp->pattern[0] &&
1066                 strcmp(fptr, pp->pattern) == 0) {
1067                 /* matched */
1068                 break;
1069             }
1070         }
1071     } else {
1072         for (pp = patterns; pp; pp = pp->next) {
1073             if (strstr(fptr, pp->pattern) != NULL) {
1074                 /* matched */
1075                 break;
1076             }
1077         }
1078     }
1079 } else {
1080     /* grep or egrep */
1081     for (pp = patterns; pp; pp = pp->next) {
1082         int rv;
1083
1084         rv = regexec(&pp->re, ptr, 0, NULL, 0);
1085         if (rv == REG_OK) {
1086             /* matched */
1087             break;
1088         }
1089     }
1090
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 }
1109
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) {
1137             if (outfn) {
1138                 (void) printf("%s:", fn);
1139             }
1140             if (bflag) {
1141                 (void) printf("%lld:", (offset_t)
1142                     (line_offset / BSIZE));
1143             }
1144             if (nflag) {
1145                 (void) printf("%lld:", lineno);
1146             }
1147             *ptrend = '\n';
1148             (void) fwrite(ptr, 1, line_len + 1, stdout);
1149         }
1150         if (ferror(stdout)) {
1151             return(0);
1152         }
1153     }
1154     L_skip_line:
1155     if (!newlinep)
1156         break;
1157     data_len -= line_len + 1;
1158     line_offset += line_len + 1;
1159     ptr = ptrend + 1;
1160 }
1161
1162 if (cflag) {
1163     if (Hflag || outfn) {
1164         if (outfn) {
1165             (void) printf("%s:", fn);
1166         }
1167         if (!qflag) {
1168             (void) printf("%lld\n", matches);
1169         }
1170     }
1171     return (matches != 0);
1172 }
1173
1174 /*
1175  * usage message for grep
1176  */
1177 static void
1178 usage(void)
1179 {
1180     if (egrep || fgrep) {
1181         (void) fprintf(stderr, gettext("Usage: %s"), cmdname);

```

```

1181         (void) fprintf(stderr,
1182             gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvx] "
1183                 gettext(" [-c|-l|-q] [-r|-R] [-bhinsvx] "
1184                     "pattern_list [file ...]\n"));
1185
1186         (void) fprintf(stderr, "\t%s", cmdname);
1187         (void) fprintf(stderr,
1188             gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvx] "
1189                 gettext(" [-c|-l|-q] [-r|-R] [-bhinsvx] "
1190                     "[-e pattern_list]... "
1191                     "[-f pattern_file]... [file...]\n"));
1192     } else {
1193         (void) fprintf(stderr, gettext("Usage:\t%s"), cmdname);
1194         (void) fprintf(stderr,
1195             gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvwx] "
1196                 gettext(" [-c|-l|-q] [-r|-R] [-bhinsvwx] "
1197                     "pattern_list [file ...]\n"));
1198
1199         (void) fprintf(stderr, "\t%s", cmdname);
1200         (void) fprintf(stderr,
1201             gettext(" [-c|-l|-q] [-r|-R] [-bhHinsvwx] "
1202                 gettext(" [-c|-l|-q] [-r|-R] [-bhinsvwx] "
1203                     "[-e pattern_list]... "
1204                     "[-f pattern_file]... [file...]\n"));
1205
1206         (void) fprintf(stderr, "\t%s", cmdname);
1207         (void) fprintf(stderr,
1208             gettext(" -E [-c|-l|-q] [-r|-R] [-bhHinsvx] "
1209                 gettext(" -E [-c|-l|-q] [-r|-R] [-bhinsvx] "
1210                     "pattern_list [file ...]\n"));
1211
1212         (void) fprintf(stderr, "\t%s", cmdname);
1213         (void) fprintf(stderr,
1214             gettext(" -E [-c|-l|-q] [-r|-R] [-bhHinsvx] "
1215                 gettext(" -E [-c|-l|-q] [-r|-R] [-bhinsvx] "
1216                     "[-e pattern_list]... "
1217                     "[-f pattern_file]... [file...]\n"));
1218
1219         (void) fprintf(stderr, "\t%s", cmdname);
1220         (void) fprintf(stderr,
1221             gettext(" -F [-c|-l|-q] [-r|-R] [-bhHinsvx] "
1222                 gettext(" -F [-c|-l|-q] [-r|-R] [-bhinsvx] "
1223                     "pattern_list [file ...]\n"));
1224
1225         (void) fprintf(stderr, "\t%s", cmdname);
1226         (void) fprintf(stderr,
1227             gettext(" -F [-c|-l|-q] [-bhHinsvx] [-e pattern_list]... "
1228                 gettext(" -F [-c|-l|-q] [-bhinsvx] [-e pattern_list]... "
1229                     "[-f pattern_file]... [file...]\n"));
1230     }
1231     exit(2);
1232     /* NOTREACHED */
1233 }

```

unchanged\_portion\_omitted

```

*****
9118 Thu Sep 12 13:22:55 2013
new/usr/src/man/man1/egrep.1
3737 grep does not support -H option
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
*****
1  \" te
2  .\" Copyright 1989 AT&T
3  .\" Copyright (c) 2006, Sun Microsystems, Inc. All Rights Reserved
4  .\" Portions Copyright (c) 1992, X/Open Company Limited All Rights Reserved
5  .\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
6  .\" http://www.opengroup.org/bookstore/.
7  .\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
8  .\" This notice shall appear on any product containing this material.
9  .\" The contents of this file are subject to the terms of the Common Development
10 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
11 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
12 .TH EGREP 1 \"May 3, 2013\"
12 .TH EGREP 1 \"Mar 24, 2006\"
13 .SH NAME
14 egrep \- search a file for a pattern using full regular expressions
15 .SH SYNOPSIS
16 .LP
17 .nf
18 \fB/usr/bin/egrep\fR [\fB-bcHhlnqsv\fR] \fB-e\fR \fIpattern_list\fR [\fIfile...
18 \fB/usr/bin/egrep\fR [\fB-bchilnsv\fR] \fB-e\fR \fIpattern_list\fR [\fIfile...f
19 .fi

21 .LP
22 .nf
23 \fB/usr/bin/egrep\fR [\fB-bcHhlnqsv\fR] \fB-f\fR \fIfile\fR [\fIfile...fR]
23 \fB/usr/bin/egrep\fR [\fB-bchilnsv\fR] \fB-f\fR \fIfile\fR [\fIfile...fR]
24 .fi

26 .LP
27 .nf
28 \fB/usr/bin/egrep\fR [\fB-bcHhlnqsv\fR] \fIpattern\fR [\fIfile...fR]
28 \fB/usr/bin/egrep\fR [\fB-bchilnsv\fR] \fIpattern\fR [\fIfile...fR]
29 .fi

31 .LP
32 .nf
33 \fB/usr/xpg4/bin/egrep\fR [\fB-bcHhlnqsvx\fR] \fB-e\fR \fIpattern_list\fR [\fB-
33 \fB/usr/xpg4/bin/egrep\fR [\fB-bchilnqsvx\fR] \fB-e\fR \fIpattern_list\fR [\fB-f
34 [\fIfile...fR]
35 .fi

37 .LP
38 .nf
39 \fB/usr/xpg4/bin/egrep\fR [\fB-bcHhlnqsvx\fR] [\fB-e\fR \fIpattern_list\fR] \fB
39 \fB/usr/xpg4/bin/egrep\fR [\fB-bchilnqsvx\fR] [\fB-e\fR \fIpattern_list\fR] \fB-
40 [\fIfile...fR]
41 .fi

43 .LP
44 .nf
45 \fB/usr/xpg4/bin/egrep\fR [\fB-bcHhlnqsvx\fR] \fIpattern\fR [\fIfile...fR]
45 \fB/usr/xpg4/bin/egrep\fR [\fB-bchilnqsvx\fR] \fIpattern\fR [\fIfile...fR]
46 .fi

48 .SH DESCRIPTION
49 .sp
50 .LP
51 The \fBgrep\fR (\fIexpression grep\fR) utility searches files for a pattern of
52 characters and prints all lines that contain that pattern. \fBgrep\fR uses
53 full regular expressions (expressions that have string values that use the full

```

```

54 set of alphanumeric and special characters) to match the patterns. It uses a
55 fast deterministic algorithm that sometimes needs exponential space.
56 .sp
57 .LP
58 If no files are specified, \fBgrep\fR assumes standard input. Normally, each
59 line found is copied to the standard output. The file name is printed before
60 each line found if there is more than one input file.
61 .SS \"/usr/bin/egrep\"
62 .sp
63 .LP
64 The \fB/usr/bin/egrep\fR utility accepts full regular expressions as described
65 on the \fBregexp\fR(5) manual page, except for \fB\(\fR and \fB\)\fR,
66 \fB\(\fR and \fB\)\fR, \fB\(\fR and \fB\)\fR, \fB\(\fR and \fB\)\fR, \fB\(\fR and \fB\)\fR, and
67 \fB\en\fR, and with the addition of:
68 .RS +4
69 .TP
70 1.
71 A full regular expression followed by \fB+\fR that matches one or more
72 occurrences of the full regular expression.
73 .RE
74 .RS +4
75 .TP
76 2.
77 A full regular expression followed by \fB?\fR that matches 0 or 1
78 occurrences of the full regular expression.
79 .RE
80 .RS +4
81 .TP
82 3.
83 Full regular expressions separated by | or by a \fBNEWLINE\fR that match
84 strings that are matched by any of the expressions.
85 .RE
86 .RS +4
87 .TP
88 4.
89 A full regular expression that can be enclosed in parentheses \fB()\fR for
90 grouping.
91 .RE
92 .sp
93 .LP
94 Be careful using the characters \fB$\fR, \fB*\fR, \fB[\fR, \fB^\fR, |, \fB\(\fR,
95 \fB)\fR, and \fB\)\fR in \fIfull regular expression\fR, because they are also
96 meaningful to the shell. It is safest to enclose the entire \fIfull regular
97 expression\fR in single quotes (\fBa'\fR\fBa'\fR).
98 .sp
99 .LP
100 The order of precedence of operators is \fB[\fR, then \fB*\fR, then \fB\(\fR, then
101 concatenation, then | and NEWLINE.
102 .SS \"/usr/xpg4/bin/egrep\"
103 .sp
104 .LP
105 The \fB/usr/xpg4/bin/egrep\fR utility uses the regular expressions described in
106 the \fBEXTENDED REGULAR EXPRESSIONS\fR section of the \fBregex\fR(5) manual
107 page.
108 .SH OPTIONS
109 .sp
110 .LP
111 The following options are supported for both \fB/usr/bin/egrep\fR and
112 \fB/usr/xpg4/bin/egrep\fR:
113 .sp
114 .ne 2
115 .na
116 \fB\b\fR
117 .ad
118 .RS 19n
119 Precede each line by the block number on which it was found. This can be useful

```

```

120 in locating block numbers by context (first block is 0).
121 .RE

123 .sp
124 .ne 2
125 .na
126 \fB\fB-c\fR\fR
127 .ad
128 .RS 19n
129 Print only a count of the lines that contain the pattern.
130 .RE

132 .sp
133 .ne 2
134 .na
135 \fB\fB-e\fR \fIpattern_list\fR
136 .ad
137 .RS 19n
138 Search for a \fIpattern_list\fR (\fIfull regular expression\fR that begins with
139 a \fB(mi\fR).
140 .RE

142 .sp
143 .ne 2
144 .na
145 \fB\fB-f\fR \fIfile\fR
146 .ad
147 .RS 19n
148 Take the list of \fIfull\fR \fIregular\fR \fIexpressions\fR from \fIfile\fR.
149 .RE

151 .sp
152 .ne 2
153 .na
154 \fB\fB-H\fR\fR
155 .ad
156 .RS 19n
157 Precedes each line by the name of the file containing the matching line.
158 .RE

160 .sp
161 .ne 2
162 .na
163 \fB\fB-h\fR\fR
164 .ad
165 .RS 19n
166 Suppress printing of filenames when searching multiple files.
167 .RE

169 .sp
170 .ne 2
171 .na
172 \fB\fB-i\fR\fR
173 .ad
174 .RS 19n
175 Ignore upper/lower case distinction during comparisons.
176 .RE

178 .sp
179 .ne 2
180 .na
181 \fB\fB-l\fR\fR
182 .ad
183 .RS 19n
184 Print the names of files with matching lines once, separated by NEWLINES. Does
185 not repeat the names of files when the pattern is found more than once.

```

```

186 .RE

188 .sp
189 .ne 2
190 .na
191 \fB\fB-n\fR\fR
192 .ad
193 .RS 19n
194 Precede each line by its line number in the file (first line is 1).
195 .RE

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

207 .sp
208 .ne 2
209 .na
210 \fB\fB-s\fR\fR
211 .ad
212 .RS 19n
213 Legacy equivalent of \fB-q\fR.
214 Work silently, that is, display nothing except error messages. This is useful
215 for checking the error status.
214 .RE

216 .sp
217 .ne 2
218 .na
219 \fB\fB-v\fR\fR
220 .ad
221 .RS 19n
222 Print all lines except those that contain the pattern.
223 .RE

225 .SS "/usr/xpg4/bin/egrep"
226 .sp
227 .LP
228 The following options are supported for \fB/usr/xpg4/bin/egrep\fR only:
229 .sp
230 .ne 2
231 .na
232 \fB\fB-q\fR\fR
233 .ad
234 .RS 6n
235 Consider only input lines that use all characters in the line to match an
236 entire fixed string or regular expression to be matching lines.
237 .RE

239 .SH OPERANDS

```

```

240 .sp
241 .LP
242 The following operands are supported:
243 .sp
244 .ne 2
245 .na
246 \fB\fifile\fR\fR
247 .ad
248 .RS 8n
249 A path name of a file to be searched for the patterns. If no \fifile\fR
250 operands are specified, the standard input is used.
251 .RE

253 .SS "/usr/bin/egrep"
254 .sp
255 .ne 2
256 .na
257 \fB\fipattern\fR\fR
258 .ad
259 .RS 11n
260 Specify a pattern to be used during the search for input.
261 .RE

263 .SS "/usr/xpg4/bin/egrep"
264 .sp
265 .ne 2
266 .na
267 \fB\fipattern\fR\fR
268 .ad
269 .RS 11n
270 Specify one or more patterns to be used during the search for input. This
271 operand is treated as if it were specified as \fB-e\fR\fR\fR\fR\fR\fR.
272 .RE

274 .SH USAGE
275 .sp
276 .LP
277 See \fBlargefile\fR(5) for the description of the behavior of \fBgrep\fR when
278 encountering files greater than or equal to 2 Gbyte ( 231 bytes).
279 .SH ENVIRONMENT VARIABLES
280 .sp
281 .LP
282 See \fBenviron\fR(5) for descriptions of the following environment variables
283 that affect the execution of \fBgrep\fR: \fBLC_COLLATE\fR, \fBLC_CTYPE\fR,
284 \fBLC_MESSAGES\fR, and \fBNLSPATH\fR.
285 .SH EXIT STATUS
286 .sp
287 .LP
288 The following exit values are returned:
289 .sp
290 .ne 2
291 .na
292 \fB\B0\fR\fR
293 .ad
294 .RS 5n
295 If any matches are found.
296 .RE

298 .sp
299 .ne 2
300 .na
301 \fB\B1\fR\fR
302 .ad
303 .RS 5n
304 If no matches are found.
305 .RE

```

```

307 .sp
308 .ne 2
309 .na
310 \fB\B2\fR\fR
311 .ad
312 .RS 5n
313 For syntax errors or inaccessible files (even if matches were found).
314 .RE

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

323 .sp
324 .TS
325 box;
326 c | c
327 l | l .
328 ATTRIBUTE TYPE ATTRIBUTE VALUE
329 _
330 CSI Not Enabled
331 .TE

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

336 .sp
337 .TS
338 box;
339 c | c
340 l | l .
341 ATTRIBUTE TYPE ATTRIBUTE VALUE
342 _
343 CSI Enabled
344 .TE

346 .SH SEE ALSO
347 .sp
348 .LP
349 \fBfgrep\fR(1), \fBgrep\fR(1), \fBsed\fR(1), \fBsh\fR(1), \fBattributes\fR(5),
350 \fBenviron\fR(5), \fBlargefile\fR(5), \fBregex\fR(5), \fBregexp\fR(5),
351 \fBXP4\fR(5)
352 .SH NOTES
353 .sp
354 .LP
355 Ideally there should be only one \fBgrep\fR command, but there is not a single
356 algorithm that spans a wide enough range of space-time trade-offs.
357 .sp
358 .LP
359 Lines are limited only by the size of the available virtual memory.
360 .SS "/usr/xpg4/bin/egrep"
361 .sp
362 .LP
363 The \fB/usr/xpg4/bin/egrep\fR utility is identical to \fB/usr/xpg4/bin/grep\fR
364 \fB-E\fR. See \fBgrep\fR(1). Portable applications should use
365 \fB/usr/xpg4/bin/grep\fR \fB-E\fR.

```

```

*****
7742 Thu Sep 12 13:22:55 2013
new/usr/src/man/man1/fgrep.1
3737 grep does not support -H option
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
*****
1  \" te
2  .\" Copyright 1989 AT&T
3  .\" Copyright (c) 2006, Sun Microsystems, Inc. All Rights Reserved
4  .\" Portions Copyright (c) 1992, X/Open Company Limited All Rights Reserved
5  .\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
6  .\" http://www.opengroup.org/bookstore/.
7  .\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
8  .\" This notice shall appear on any product containing this material.
9  .\" The contents of this file are subject to the terms of the Common Development
10 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
11 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
12 .TH FGREP 1 \"May 3, 2013\"
12 .TH FGREP 1 \"Mar 24, 2006\"
13 .SH NAME
14 fgrep \- search a file for a fixed-character string
15 .SH SYNOPSIS
16 .LP
17 .nf
18 \fB/usr/bin/fgrep\fR [\fB-bcHhlnqsvx\fR] \fB-e\fR \fIpattern_list\fR [\fIfile..
18 \fB/usr/bin/fgrep\fR [\fB-bchilnsvx\fR] \fB-e\fR \fIpattern_list\fR [\fIfile...
19 .fi

21 .LP
22 .nf
23 \fB/usr/bin/fgrep\fR [\fB-bcHhlnqsvx\fR] \fB-f\fR \fIfile\fR [\fIfile...
23 \fB/usr/bin/fgrep\fR [\fB-bchilnsvx\fR] \fB-f\fR \fIfile\fR [\fIfile...
24 .fi

26 .LP
27 .nf
28 \fB/usr/bin/fgrep\fR [\fB-bcHhlnqsvx\fR] \fIpattern\fR [\fIfile...
28 \fB/usr/bin/fgrep\fR [\fB-bchilnsvx\fR] \fIpattern\fR [\fIfile...
29 .fi

31 .LP
32 .nf
33 \fB/usr/xpg4/bin/fgrep\fR [\fB-bcHhlnqsvx\fR] \fB-e\fR \fIpattern_list\fR [\fB-
33 \fB/usr/xpg4/bin/fgrep\fR [\fB-bchilnqsvx\fR] \fB-e\fR \fIpattern_list\fR [\fB-f
34 [\fIfile...
35 .fi

37 .LP
38 .nf
39 \fB/usr/xpg4/bin/fgrep\fR [\fB-bcHhlnqsvx\fR] [\fB-e\fR \fIpattern_list\fR] \fB
39 \fB/usr/xpg4/bin/fgrep\fR [\fB-bchilnqsvx\fR] [\fB-e\fR \fIpattern_list\fR] \fB-
40 [\fIfile...
41 .fi

43 .LP
44 .nf
45 \fB/usr/xpg4/bin/fgrep\fR [\fB-bcHhlnqsvx\fR] \fIpattern\fR [\fIfile...
45 \fB/usr/xpg4/bin/fgrep\fR [\fB-bchilnqsvx\fR] \fIpattern\fR [\fIfile...
46 .fi

48 .SH DESCRIPTION
49 .sp
50 .LP
51 The \fBfgrep\fR (fast \fBgrep\fR) utility searches files for a character string
52 and prints all lines that contain that string. \fBfgrep\fR is different from
53 \fBgrep\fR(1) and from \fBbegrep\fR(1) because it searches for a string, instead

```

```

54 of searching for a pattern that matches an expression. \fBfgrep\fR uses a fast
55 and compact algorithm.
56 .sp
57 .LP
58 The characters \fB$\fR, \fB*\fR, \fB[\fR, \fB^\fR, \fB|\fR, \fB(\fR, \fB)\fR, and
59 \fB\efR are interpreted literally by \fBfgrep\fR, that is, \fBfgrep\fR does
60 not recognize full regular expressions as does \fBbegrep\fR. These characters
61 have special meaning to the shell. Therefore, to be safe, enclose the entire
62 \fIstring\fR within single quotes (\fB'\fR).
63 .sp
64 .LP
65 If no files are specified, \fBfgrep\fR assumes standard input. Normally, each
66 line that is found is copied to the standard output. The file name is printed
67 before each line that is found if there is more than one input file.
68 .SH OPTIONS
69 .sp
70 .LP
71 The following options are supported for both \fB/usr/bin/fgrep\fR and
72 \fB/usr/xpg4/bin/fgrep\fR:
73 .sp
74 .ne 2
75 .na
76 \fB\fB-b\fR\fR
77 .ad
78 .RS 19n
79 Precedes each line by the block number on which the line was found. This can be
80 useful in locating block numbers by context. The first block is 0.
81 .RE

83 .sp
84 .ne 2
85 .na
86 \fB\fB-c\fR\fR
87 .ad
88 .RS 19n
89 Prints only a count of the lines that contain the pattern.
90 .RE

92 .sp
93 .ne 2
94 .na
95 \fB\fB-e\fR \fIpattern_list\fR\fR
96 .ad
97 .RS 19n
98 Searches for a \fIstring\fR in \fIpattern-list\fR. This is useful when the
99 \fIstring\fR begins with a \fB(mi\fR&.
100 .RE

102 .sp
103 .ne 2
104 .na
105 \fB\fB-f\fR \fIpattern-file\fR\fR
106 .ad
107 .RS 19n
108 Takes the list of patterns from \fIpattern-file\fR.
109 .RE

111 .sp
112 .ne 2
113 .na
114 \fB\fB-H\fR\fR
115 .ad
116 .RS 19n
117 Precedes each line by the name of the file containing the matching line.
118 .RE

```



```

120 .sp
121 .ne 2
122 .na
123 \fB\fB-h\fR\fR
124 .ad
125 .RS 19n
126 Suppresses printing of files when searching multiple files.
127 .RE

129 .sp
130 .ne 2
131 .na
132 \fB\fB-i\fR\fR
133 .ad
134 .RS 19n
135 Ignores upper/lower case distinction during comparisons.
136 .RE

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

148 .sp
149 .ne 2
150 .na
151 \fB\fB-n\fR\fR
152 .ad
153 .RS 19n
154 Precedes each line by its line number in the file. The first line is 1.
155 .RE

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

167 .sp
168 .ne 2
169 .na
170 \fB\fB-s\fR\fR
161 \fB\fB-v\fR\fR
171 .ad
172 .RS 19n
173 Legacy equivalent of \fB-q\fR.
164 Prints all lines except those that contain the pattern.
174 .RE

176 .sp
177 .ne 2
178 .na
179 \fB\fB-v\fR\fR
170 \fB\fB-x\fR\fR

```

```

180 .ad
181 .RS 19n
182 Prints all lines except those that contain the pattern.
173 Prints only lines that are matched entirely.
183 .RE

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

194 .SH OPERANDS
195 .sp
196 .LP
197 The following operands are supported:
198 .sp
199 .ne 2
200 .na
201 \fB\fIfile\fR\fR
202 .ad
203 .RS 8n
204 Specifies a path name of a file to be searched for the patterns. If no
205 \fIfile\fR operands are specified, the standard input will be used.
206 .RE

208 .SS "/usr/bin/fgrep"
209 .sp
210 .ne 2
211 .na
212 \fB\fIpattern\fR\fR
213 .ad
214 .RS 11n
215 Specifies a pattern to be used during the search for input.
216 .RE

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

229 .SH USAGE
230 .sp
231 .LP
232 See \fBlargefile\fR(5) for the description of the behavior of \fBfgrep\fR when
233 encountering files greater than or equal to 2 Gbyte ( 2^31 bytes).
234 .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 \fB0\fR
248 .ad
249 .RS 5n
250 If any matches are found
251 .RE

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

262 .sp
263 .ne 2
264 .na
265 \fB2\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 \fBbed\fR(1), \fBbegrep\fR(1), \fBbgrep\fR(1), \fBbsed\fR(1), \fBbsh\fR(1),
292 \fBbattributes\fR(5), \fBbenviron\fR(5), \fBblargefile\fR(5), \fBXP4\fR(5)
293 .SH NOTES
294 .sp
295 .LP
296 Ideally, there should be only one \fBbgrep\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 Thu Sep 12 13:22:56 2013
new/usr/src/man/man1/grep.1
3737 grep does not support -H option
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
*****
1 \" te
2.\" Copyright 2012 Nexenta Systems, Inc. All rights reserved.
3.\" Copyright 1989 AT&T
4.\" Copyright (c) 2008, Sun Microsystems, Inc. All Rights Reserved
5.\" Portions Copyright (c) 1992, X/Open Company Limited All Rights Reserved
6.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
7.\" http://www.opengroup.org/bookstore/.
8.\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
9.\" This notice shall appear on any product containing this material.
10.\" The contents of this file are subject to the terms of the Common Development
11.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
12.\" When distributing Covered Code, include this CDDL HEADER in each file and in
13.TH GREP 1 \"May 3, 2013\"
13.TH GREP 1 \"Feb 26, 2008\"
14.SH NAME
15.grep \- search a file for a pattern
16.SH SYNOPSIS
17.LP
18.nf
19.\fB/usr/bin/grep\fR [\fB-c\fR | \fB-l\fR | \fB-q\fR] [\fB-r\fR | \fB-R\fR] [\fB-b
19.\fB/usr/bin/grep\fR [\fB-c\fR | \fB-l\fR | \fB-q\fR] [\fB-r\fR | \fB-R\fR] [\fB-b
20.\fB-limited-regular-expression\fR [\fBfilename\fR]...
21.fi

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

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

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

43.SH DESCRIPTION
44.sp
45.LP
46.The \fBgrep\fR utility searches text files for a pattern and prints all lines
47.that contain that pattern. It uses a compact non-deterministic algorithm.
48.sp
49.LP
50.Be careful using the characters \fB$\fR, \fB*\fR, \fB[\fR, \fB^\fR, \fB|\fR,
51.\fB\fR, \fB)\fR, and \fB)\fR in the \fIpattern_list\fR because they are also
52.meaningful to the shell. It is safest to enclose the entire \fIpattern_list\fR
53.in single quotes \fBa'\fR&...\fBa'\fR&.
54.sp
55.LP

```

```

56.If no files are specified, \fBgrep\fR assumes standard input. Normally, each
57.line found is copied to standard output. The file name is printed before each
58.line found if there is more than one input file.
59.SS \"/usr/bin/grep\"
60.sp
61.LP
62.The \fB/usr/bin/grep\fR utility uses limited regular expressions like those
63.described on the \fBregex\fR(5) manual page to match the patterns.
64.SS \"/usr/xpg4/bin/grep\"
65.sp
66.LP
67.The options \fB-E\fR and \fB-F\fR affect the way \fB/usr/xpg4/bin/grep\fR
68.interprets \fIpattern_list\fR. If \fB-E\fR is specified,
69.\fB/usr/xpg4/bin/grep\fR interprets \fIpattern_list\fR as a full regular
70.expression (see \fB-E\fR for description). If \fB-F\fR is specified,
71.\fBgrep\fR interprets \fIpattern_list\fR as a fixed string. If neither are
72.specified, \fBgrep\fR interprets \fIpattern_list\fR as a basic regular
73.expression as described on \fBregex\fR(5) manual page.
74.SH OPTIONS
75.sp
76.LP
77.The following options are supported for both \fB/usr/bin/grep\fR and
78.\fB/usr/xpg4/bin/grep\fR:
79.sp
80.ne 2
81.na
82.\fB\b\b\fR
83.ad
84.RS 6n
85.Precedes each line by the block number on which it was found. This can be
86.useful in locating block numbers by context (first block is 0).
87.RE

89.sp
90.ne 2
91.na
92.\fB\b-c\fR
93.ad
94.RS 6n
95.Prints only a count of the lines that contain the pattern.
96.RE

98.sp
99.ne 2
100.na
101.\fB\b-H\fR
102.ad
103.RS 6n
104.Precedes each line by the name of the file containing the matching line.
105.RE

107.sp
108.ne 2
109.na
110.\fB\b-h\fR
111.ad
112.RS 6n
113.Prevents the name of the file containing the matching line from being prepended
114.to that line. Used when searching multiple files.
115.RE

117.sp
118.ne 2
119.na
120.\fB\b-i\fR
121.ad

```

```

122 .RS 6n
123 Ignores upper/lower case distinction during comparisons.
124 .RE

126 .sp
127 .ne 2
128 .na
129 \fB\fB-l\fR\fR
130 .ad
131 .RS 6n
132 Prints only the names of files with matching lines, separated by NEWLINE
133 characters. Does not repeat the names of files when the pattern is found more
134 than once.
135 .RE

137 .sp
138 .ne 2
139 .na
140 \fB\fB-n\fR\fR
141 .ad
142 .RS 6n
143 Precedes each line by its line number in the file (first line is 1).
144 .RE

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

157 .sp
158 .ne 2
159 .na
160 \fB\fB-R\fR\fR
161 .ad
162 .RS 6n
163 Read all files under each directory, recursively, following all symbolic links.
164 .RE

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

176 .sp
177 .ne 2
178 .na
179 \fB\fB-s\fR\fR
180 .ad
181 .RS 6n
182 Suppresses error messages about nonexistent or unreadable files.
183 .RE

185 .sp
186 .ne 2
187 .na

```

```

188 \fB\fB-v\fR\fR
189 .ad
190 .RS 6n
191 Prints all lines except those that contain the pattern.
192 .RE

194 .sp
195 .ne 2
196 .na
197 \fB\fB-w\fR\fR
198 .ad
199 .RS 6n
200 Searches for the expression as a word as if surrounded by \fB\e<\fR and
201 \fB\e>\fR\&.
202 .RE

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

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

```

```

254 .TP
255 4.
256 A full regular expression that is enclosed in parentheses \fB()\fR for
257 grouping.
258 .RE
259 The order of precedence of operators is \fB[|\]\fR, then \fB*\|?\|+\fR, then
260 concatenation, then | and new-line.
261 .RE

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

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

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

298 .SH OPERANDS
299 .sp
300 .LP
301 The following operands are supported:
302 .sp
303 .ne 2
304 .na
305 \fB\fB-f\fR \fIfile\fR
306 .ad
307 .RS 8n
308 A path name of a file to be searched for the patterns. If no \fIfile\fR
309 operands are specified, the standard input is used.
310 .RE

312 .SS "/usr/bin/grep"
313 .sp
314 .ne 2
315 .na
316 \fB\fB-f\fR \fIpattern\fR\fR
317 .ad
318 .RS 11n
319 Specifies a pattern to be used during the search for input.

```

```

320 .RE

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

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

368 .sp
369 .in +2
370 .nf
371 example% \fB/usr/bin/grep -i -n posix text.mm\fR
372 .fi
373 .in -2
374 .sp

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

382 .sp
383 .in +2
384 .nf
385 example% \fB/usr/bin/grep ^$\fR

```

```

386 .fi
387 .in -2
388 .sp

390 .sp
391 .LP
392 or

394 .sp
395 .in +2
396 .nf
397 example% \fB/usr/bin/grep -v \fR
398 .fi
399 .in -2
400 .sp

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

409 .sp
410 .in +2
411 .nf
412 example% \fB/usr/xpg4/bin/grep 'abc
413 def'\fR
414 example% \fB/usr/xpg4/bin/grep -e 'abc
415 def'\fR
416 example% \fB/usr/xpg4/bin/grep -e 'abc' -e 'def'\fR
417 example% \fB/usr/xpg4/bin/grep -E 'abc|def'\fR
418 example% \fB/usr/xpg4/bin/grep -E -e 'abc|def'\fR
419 example% \fB/usr/xpg4/bin/grep -E -e 'abc' -e 'def'\fR
420 example% \fB/usr/xpg4/bin/grep -E 'abc
421 def'\fR
422 example% \fB/usr/xpg4/bin/grep -E -e 'abc
423 def'\fR
424 example% \fB/usr/xpg4/bin/grep -F -e 'abc' -e 'def'\fR
425 example% \fB/usr/xpg4/bin/grep -F 'abc
426 def'\fR
427 example% \fB/usr/xpg4/bin/grep -F -e 'abc
428 def'\fR
429 .fi
430 .in -2
431 .sp

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

440 .sp
441 .in +2
442 .nf
443 example% \fB/usr/xpg4/bin/grep -E '^abc$ ^def$'\fR
444 example% \fB/usr/xpg4/bin/grep -F -x 'abc def'\fR
445 .fi
446 .in -2
447 .sp

449 .SH ENVIRONMENT VARIABLES
450 .sp
451 .LP

```

```

452 See \fBenviron\fR(5) for descriptions of the following environment variables
453 that affect the execution of \fBgrep\fR: \fBLANG\fR, \fBLC_ALL\fR,
454 \fBLC_COLLATE\fR, \fBLC_CTYPE\fR, \fBLC_MESSAGES\fR, and \fBLC_PATH\fR.
455 .SH EXIT STATUS
456 .sp
457 .LP
458 The following exit values are returned:
459 .sp
460 .ne 2
461 .na
462 \fB0\fR
463 .ad
464 .RS 5n
465 One or more matches were found.
466 .RE

468 .sp
469 .ne 2
470 .na
471 \fB1\fR
472 .ad
473 .RS 5n
474 No matches were found.
475 .RE

477 .sp
478 .ne 2
479 .na
480 \fB2\fR
481 .ad
482 .RS 5n
483 Syntax errors or inaccessible files (even if matches were found).
484 .RE

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

493 .sp
494 .TS
495 box;
496 c | c
497 l | l .
498 ATTRIBUTE TYPE ATTRIBUTE VALUE
499 -
500 CSI Not Enabled
501 .TE

503 .SS "/usr/xpg4/bin/grep"
504 .sp

506 .sp
507 .TS
508 box;
509 c | c
510 l | l .
511 ATTRIBUTE TYPE ATTRIBUTE VALUE
512 -
513 CSI Enabled
514 -
515 Interface Stability Committed
516 -
517 Standard See \fBstandards\fR(5).

```

```
518 .TE
520 .SH SEE ALSO
521 .sp
522 .LP
523 \fBgrep\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 \fBBLINE_MAX\fR bytes or contain binary data. \fBBLINE_MAX\fR is defined in
538 \fB/usr/include/limits.h\fR.
```