```
**********************************************************
   10185 Mon Sep 17 14:32:32 2012
new/usr/src/cmd/grep/grep.c
3047 grep support for -r would be useful
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

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

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

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

#include <sys/types.h>

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

static const char *errstr[] = {
        "Range endpoint too large.",
        "Bad number.",
```

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

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

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

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

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

        (void) setlocale(LC_ALL, "");
#if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
#define TEXT_DOMAIN "SYS_TEST"  /* Use this only if it weren't */
#endif
        (void) textdomain(TEXT_DOMAIN);
        while ((c = getopt(argc, argv, "hqblcnRrsviyw")) != -1)
        while ((c = getopt(argc, argv, "hqblcnsviyw")) != -1)
                switch (c) {
                case ’h’:
                        hflag++;
                        break;
                case ’q’:       /* POSIX: quiet: status only */
                        qflag++;
                        break;
                case ’v’:
                        vflag++;
                        break;
                case ’c’:
                        cflag++;
                        break;
                case ’n’:
                        nflag++;
                        break;
                case ’R’:
                        Rflag++;
                        /* FALLTHROUGH */
                case ’r’:
                        rflag++;
                        break;
                case ’b’:
                        bflag++;
                        break;
                case ’s’:
                        sflag++;
                        break;
                case ’l’:
                        lflag++;
                        break;
                case ’y’:
                case ’i’:
                        iflag++;
                        break;
                case ’w’:
                        wflag++;
                        break;
                case ’?’:
                        errflg++;
                }

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

        argv = &argv[optind];
        argc -= optind;
        nfile = argc - 1;

        if (strrchr(*argv, ’\n’) != NULL)
                regerr(41);

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

        if (wflag) {
                unsigned int    wordlen;
```

```
                char            *wordbuf;

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

                (void) strcpy(wordbuf, "\\<");
                (void) strcat(wordbuf, *argv);
                (void) strcat(wordbuf, "\\>");
                *argv = wordbuf;
        }

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

        if (--argc == 0)
                execute(NULL, 0);
                execute(NULL);
        else
                while (argc-- > 0)
                        prepare(*++argv);
                        execute(*++argv);

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

static void
prepare(const char *path)
execute(char *file)
{
        struct  stat st;
        int     walkflags = FTW_CHDIR;
        char    *buf = NULL;

        if (rflag) {
                if (stat(path, &st) != -1 &&
                    (st.st_mode & S_IFMT) == S_IFDIR) {
                        outfn = 1;

                        /*
                         * Add trailing slash if arg
                         * is directory, to resolve symlinks.
                         */
                        if (path[strlen(path) - 1] != ’/’) {
                                (void) asprintf(&buf, "%s/", path);
                                if (buf != NULL)
                                        path = buf;
                        }

                        /*
                         * Search through subdirs if path is directory.
                         * Don’t follow symlinks if Rflag is not set.
                         */
                        if (!Rflag)
                                walkflags |= FTW_PHYS;

                        if (nftw(path, recursive, MAX_DEPTH, walkflags) != 0) {
                                if (!sflag)
                                        errmsg("grep: can’t open %s\n", path);
                                nsucc = 2;
                        }
                        return;
                }
        }
        execute(path, 0);
}

static int
recursive(const char *name, const struct stat *statp, int info, struct FTW *ftw)
{
        /*
         * process files and follow symlinks if Rflag set.
         */
        if (info != FTW_F) {
                if (!sflag &&
                    (info == FTW_SLN || info == FTW_DNR || info == FTW_NS)) {
                        /* report broken symlinks and unreadable files */
                        errmsg("grep: can't open %s\n", name);
                }
                return (0);
        }

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

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

static void
execute(const char *file, int base)
{
        char    *lbuf, *p;
        long    count;
        long    offset = 0;
        char    *next_ptr = NULL;
        long    next_count = 0;

        tln = 0;

        if (prntbuf == NULL) {
                fw_lPrntBufLen = GBUFSIZ + 1;
                if ((prntbuf = malloc(fw_lPrntBufLen)) == NULL) {
                        exit(2); /* out of memory - BAIL */
                }
                if ((linebuf = malloc(fw_lPrntBufLen)) == NULL) {
                        exit(2); /* out of memory - BAIL */
                }
        }

        if (file == NULL)
                temp = 0;
        else if ((temp = open(file + base, O_RDONLY)) == -1) {
        else if ((temp = open(file, O_RDONLY)) == -1) {
                if (!sflag)
                        errmsg("grep: can't open %s\n", file);
                nsucc = 2;
                return;
        }

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

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

```
                        if (!rflag)
                        (void) fprintf(stdout, "%lld\n", tln);
                }
                return;
        }

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

                        /*
                         * shift unused data to the beginning of the buffer
                         */
                        if (ptr > prntbuf) {
                                (void) memmove(prntbuf, ptr, offset);
                                ptr = prntbuf;
                        }

                        /*
                         * re-allocate a larger buffer if this one is full
                         */
                        if (offset + GBUFSIZ > fw_lPrntBufLen) {
                                /*
                                 * allocate a new buffer and preserve the
                                 * contents...
                                 */
                                fw_lPrntBufLen += GBUFSIZ;
                                if ((prntbuf = realloc(prntbuf,
                                    fw_lPrntBufLen)) == NULL)
                                        exit(2);

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

                                ptr = prntbuf;
                        }

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

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

                        /* last line of file has no newline */
                        ptrend = ptr + offset;
                        nlflag = 0;
                } else {
                        next_ptr = ptrend + 1;
                        next_count = offset + count - (next_ptr - ptr);
                        nlflag = 1;
                }
                lnum++;
                *ptrend = '\0';

                if (iflag) {
                        /*
                         * Make a lower case copy of the record
                         */
                        p = ptr;
                        for (lbuf = linebuf; p < ptrend; )
                                *lbuf++ = (char)tolower((int)
                                    (unsigned char)*p++);
                        *lbuf = '\0';
                        lbuf = linebuf;
                } else
                        /*
                         * Use record as is
                         */
                        lbuf = ptr;

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

                if (!nlflag)
                        break;

                ptr = next_ptr;
                count = next_count;
                offset = 0;
        }
        (void) close(temp);

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

static int
succeed(const char *f)
succeed(char *f)
{
        int nchars;
        nsucc = (nsucc == 2) ? 2 : 1;

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

        if (qflag) {
                /* no need to continue */
                return (1);
        }

        if (cflag) {
                tln++;
                return (0);
        }

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

        if (!hflag && (nfile > 1 || (rflag && outfn))) {
        if (nfile > 1 && !hflag)
                /* print filename */
                (void) fprintf(stdout, "%s:", f);
```

```
        }

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

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

        if (nlflag) {
                /* newline at end of line */
                *ptrend = '\n';
                nchars = ptrend - ptr + 1;
        } else {
                /* don't write sentinel \0 */
                nchars = ptrend - ptr;
        }

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

______***unchanged_portion_omitted_***

```
**********************************************************
   27925 Mon Sep 17 14:32:33 2012
new/usr/src/cmd/grep_xpg4/grep.c
3047 grep support for -r would be useful
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

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

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

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

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

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

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

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

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

static int      use_wchar, use_bmg, mblocale;

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

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

/*
 * mainline for grep
 */
int
main(int argc, char **argv)
{
        char    *ap;
        int     matched = 0;
        int     c;
        int     fflag = 0;
        int     errors = 0;
        int     i, n_pattern = 0, n_file = 0;
        char    **pattern_list = NULL;
        char    **file_list = NULL;

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

        /*
         * true if this is running on the multibyte locale
         */
        mblocale = (MB_CUR_MAX > 1);
        /*
         * Skip leading slashes
         */
        cmdname = argv[0];
        if (ap = strrchr(cmdname, ’/’))
                cmdname = ap + 1;

        ap = cmdname;
        /*
         * Detect egrep/fgrep via command name, map to -E and -F options.
         */
        if (*ap == ’e’ || *ap == ’E’) {
                regflags |= REG_EXTENDED;
                egrep++;
        } else {
                if (*ap == ’f’ || *ap == ’F’) {
                        fgrep++;
                }
        }

        while ((c = getopt(argc, argv, "vwchilnrbse:f:qxEFIR")) != EOF) {
        while ((c = getopt(argc, argv, "vwchilnbse:f:qxEFI")) != EOF) {
                switch (c) {
                case ’v’:       /* POSIX: negate matches */
                        nvflag = 0;
                        break;

                case ’c’:       /* POSIX: write count */
                        cflag++;
                        break;

                case ’i’:       /* POSIX: ignore case */
                        iflag++;
                        regflags |= REG_ICASE;
                        break;

                case ’l’:       /* POSIX: Write filenames only */
                        lflag++;
                        break;

                case ’n’:       /* POSIX: Write line numbers */
                        nflag++;
                        break;

                case ’r’:       /* Solaris: search recursively */
                        rflag++;
                        break;

                case ’b’:       /* Solaris: Write file block numbers */
                        bflag++;
```

```
                        break;

                case ’s’:       /* POSIX: No error msgs for files */
                        sflag++;
                        break;

                case ’e’:       /* POSIX: pattern list */
                        n_pattern++;
                        pattern_list = realloc(pattern_list,
                            sizeof (char *) * n_pattern);
                        if (pattern_list == NULL) {
                                (void) fprintf(stderr,
                                    gettext("%s: out of memory\n"),
                                    cmdname);
                                exit(2);
                        }
                        *(pattern_list + n_pattern - 1) = optarg;
                        break;

                case ’f’:       /* POSIX: pattern file */
                        fflag = 1;
                        n_file++;
                        file_list = realloc(file_list,
                            sizeof (char *) * n_file);
                        if (file_list == NULL) {
                                (void) fprintf(stderr,
                                    gettext("%s: out of memory\n"),
                                    cmdname);
                                exit(2);
                        }
                        *(file_list + n_file - 1) = optarg;
                        break;
                case ’h’:       /* Solaris: supress printing of file name */
                        hflag = 1;
                        break;

                case ’q’:       /* POSIX: quiet: status only */
                        qflag++;
                        break;

                case ’w’:       /* Solaris: treat pattern as word */
                        wflag++;
                        break;

                case ’x’:       /* POSIX: full line matches */
                        xflag++;
                        regflags |= REG_ANCHOR;
                        break;

                case ’E’:       /* POSIX: Extended RE’s */
                        regflags |= REG_EXTENDED;
                        Eflag++;
                        break;

                case ’F’:       /* POSIX: strings, not RE’s */
                        Fflag++;
                        break;

                case ’R’:       /* Solaris: like rflag, but follow symlinks */
                        Rflag++;
                        rflag++;
                        break;

                default:
                        usage();
                }
```

```
        }
        /*
         * If we’re invoked as egrep or fgrep we need to do some checks
         */

        if (egrep || fgrep) {
                /*
                 * Use of -E or -F with egrep or fgrep is illegal
                 */
                if (Eflag || Fflag)
                        usage();
                /*
                 * Don’t allow use of wflag with egrep / fgrep
                 */
                if (wflag)
                        usage();
                /*
                 * For Solaris the -s flag is equivalent to XCU -q
                 */
                if (sflag)
                        qflag++;
                /*
                 * done with above checks - set the appropriate flags
                 */
                if (egrep)
                        Eflag++;
                else                    /* Else fgrep */
                        Fflag++;
        }

        if (wflag && (Eflag || Fflag)) {
                /*
                 * -w cannot be specified with grep -F
                 */
                usage();
        }

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

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

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

        /*
         * Now handling -e and -f option
         */
        if (pattern_list) {
                for (i = 0; i < n_pattern; i++) {
                        addpattern(pattern_list[i]);
                }
                free(pattern_list);
        }
        if (file_list) {
                for (i = 0; i < n_file; i++) {
                        addfile(file_list[i]);
```

```
                }
                free(file_list);
        }

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

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

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

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

                        if ((fd = open(*argv, O_RDONLY)) == -1) {
                                errors = 1;
                                if (sflag)
                                        continue;
                                (void) fprintf(stderr, gettext(
                                    "%s: can’t open \"%s\"\n"),
                                    cmdname, *argv);
                                continue;
                }
                        matched |= grep(fd, *argv);
                        (void) close(fd);
                        if (ferror(stdout))
                                break;
        }
        }
        /*
         * Return() here is used instead of exit
         */

        (void) fflush(stdout);

        if (errors)
                return (2);
        return (matched ? 0 : 1);
}

static void
process_path(const char *path)
```

```c
{
        struct  stat st;
        int     walkflags = FTW_CHDIR;
        char    *buf = NULL;

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

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

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

                        if (nftw(path, recursive, MAX_DEPTH, walkflags) != 0) {
                                if (!sflag)
                                        (void) fprintf(stderr,
                                            gettext("%s: can't open \"%s\"\n"),
                                            cmdname, path);
                                errors = 1;
                        }
                        return;
                }
        }
        process_file(path, 0);
}

/*
 * Read and process all files in directory recursively.
 */
static int
recursive(const char *name, const struct stat *statp, int info, struct FTW *ftw)
{
        /*
         * Process files and follow symlinks if Rflag set.
         */
        if (info != FTW_F) {
                /* Report broken symlinks and unreadable files */
                if (!sflag &&
                    (info == FTW_SLN || info == FTW_DNR || info == FTW_NS)) {
                        (void) fprintf(stderr,
                            gettext("%s: can't open \"%s\"\n"), cmdname, name);
                }
                return (0);
        }


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

```c
}

/*
 * Opens file and call grep function.
 */
static void
process_file(const char *name, int base)
{
        int fd;

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

        if (ferror(stdout)) {
                (void) fprintf(stderr, gettext(
                    "%s: error writing to stdout\n"),
                    cmdname);
                (void) fflush(stdout);
                exit(2);
        }

}

/*
 * Add a file of strings to the pattern list.
 */
static void
addfile(const char *fn)
addfile(char *fn)
{
        FILE    *fp;
        char    *inbuf;
        char    *bufp;
        size_t  bufsiz, buflen, bufused;

        /*
         * Open the pattern file
         */
        if ((fp = fopen(fn, "r")) == NULL) {
                (void) fprintf(stderr, gettext("%s: can't open \"%s\"\n"),
                    cmdname, fn);
                exit(2);
        }
        bufsiz = BUFSIZE;
        if ((inbuf = malloc(bufsiz)) == NULL) {
                (void) fprintf(stderr,
                    gettext("%s: out of memory\n"), cmdname);
                exit(2);
        }
        bufp = inbuf;
        bufused = 0;
        /*
         * Read in the file, reallocing as we need more memory
         */
        while (fgets(bufp, bufsiz - bufused, fp) != NULL) {
                buflen = strlen(bufp);
                bufused += buflen;
                if (bufused + 1 == bufsiz && bufp[buflen - 1] != '\n') {
                        /*
                         * if this line does not fit to the buffer,
                         * realloc larger buffer
                         */
                        bufsiz += BUFSIZE;
                        if ((inbuf = realloc(inbuf, bufsiz)) == NULL) {
                                (void) fprintf(stderr,
                                    gettext("%s: out of memory\n"),
                                    cmdname);
                                exit(2);
                        }
                        bufp = inbuf + bufused;
                        continue;
                }
                if (bufp[buflen - 1] == ’\n’) {
                        bufp[--buflen] = ’\0’;
                }
                addpattern(inbuf);

                bufp = inbuf;
                bufused = 0;
        }
        free(inbuf);
        (void) fclose(fp);
}
______unchanged_portion_omitted_

/*
 * Do grep on a single file.
 * Return true in any lines matched.
 *
 * We have two strategies:
 * The fast one is used when we have a single pattern with
 * a string known to occur in the pattern. We can then
 * do a BMG match on the whole buffer.
 * This is an order of magnitude faster.
 * Otherwise we split the buffer into lines,
 * and check for a match on each line.
 */
static int
grep(int fd, const char *fn)
grep(int fd, char *fn)
{
        PATTERN *pp;
        off_t   data_len;       /* length of the data chunk */
        off_t   line_len;       /* length of the current line */
        off_t   line_offset;    /* current line’s offset from the beginning */
        long long       lineno;
        long long       matches = 0;    /* Number of matching lines */
        int     newlinep;       /* 0 if the last line of file has no newline */
        char    *ptr, *ptrend;


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

        pp = patterns;

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

        if (use_wchar && outline == NULL) {
                outbuflen = BUFSIZE + 1;
                outline = malloc(sizeof (wchar_t) * outbuflen);
                if (outline == NULL) {
```

```
                        (void) fprintf(stderr, gettext("%s: out of memory\n"),
                            cmdname);
                        exit(2);
                }
        }

        if (prntbuf == NULL) {
                prntbuflen = BUFSIZE;
                if ((prntbuf = malloc(prntbuflen + 1)) == NULL) {
                        (void) fprintf(stderr, gettext("%s: out of memory\n"),
                            cmdname);
                        exit(2);
                }
        }

        line_offset = 0;
        lineno = 0;
        newlinep = 1;
        data_len = 0;
        for (; ; ) {
                long    count;
                off_t   offset = 0;

                if (data_len == 0) {
                        /*
                         * If no data in the buffer, reset ptr
                         */
                        ptr = prntbuf;
                }
                if (ptr == prntbuf) {
                        /*
                         * The current data chunk starts from prntbuf.
                         * This means either the buffer has no data
                         * or the buffer has no newline.
                         * So, read more data from input.
                         */
                        count = read(fd, ptr + data_len, prntbuflen - data_len);
                        if (count < 0) {
                                /* read error */
                                if (cflag) {
                                        if (outfn && !rflag) {
                                        if (outfn) {
                                                (void) fprintf(stdout,
                                                    "%s:", fn);
                                        }
                                        if (!qflag && !rflag) {
                                        if (!qflag) {
                                                (void) fprintf(stdout, "%lld\n",
                                                    matches);
                                        }
                                }
                                return (0);
                        } else if (count == 0) {
                                /* no new data */
                                if (data_len == 0) {
                                        /* end of file already reached */
                                        break;
                                }
                                /* last line of file has no newline */
                                ptrend = ptr + data_len;
                                newlinep = 0;
                                goto L_start_process;
                        }
                        offset = data_len;
                        data_len += count;
                }
                /*
                 * Look for newline in the chunk
                 * between ptr + offset and ptr + data_len - offset.
                 */
                ptrend = find_nl(ptr + offset, data_len - offset);
                if (ptrend == NULL) {
                        /* no newline found in this chunk */
                        if (ptr > prntbuf) {
                                /*
                                 * Move remaining data to the beginning
                                 * of the buffer.
                                 * Remaining data lie from ptr for
                                 * data_len bytes.
                                 */
                                (void) memmove(prntbuf, ptr, data_len);
                        }
                        if (data_len == prntbuflen) {
                                /*
                                 * No enough room in the buffer
                                 */
                                prntbuflen += BUFSIZE;
                                prntbuf = realloc(prntbuf, prntbuflen + 1);
                                if (prntbuf == NULL) {
                                        (void) fprintf(stderr,
                                            gettext("%s: out of memory\n"),
                                            cmdname);
                                        exit(2);
                                }
                        }
                        ptr = prntbuf;
                        /* read the next input */
                        continue;
                }
L_start_process:

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

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

                                /*
                                 * When this chunk does not contain newline,
                                 * ptrend becomes NULL, which should happen
                                 * when the last line of file does not end
                                 * with a newline.  At such a point,
                                 * newlinep should have been set to 0.
```

```
                                 * Therefore, just after jumping to
                                 * L_skip_line, the main for-loop quits,
                                 * and the line_len value won’t be
                                 * used.
                                 */
                                line_len = ptrend - ptr;
                                goto L_skip_line;
                        }
                        if (bline > ptrend) {
                                /*
                                 * Pattern found not in the first line
                                 * of this chunk.
                                 * Discard the first line.
                                 */
                                line_len = ptrend - ptr;
                                goto L_skip_line;
                        }
                        /*
                         * Pattern found in the first line of this chunk.
                         * Using this result.
                         */
                        *ptrend = ’\0’;
                        line_len = ptrend - ptr;

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

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

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

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

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

                        if (xflag) {
                                for (pp = patterns; pp; pp = pp->next) {
                                        if (outline[0] == pp->wpattern[0] &&
                                            wcscmp(outline,
                                            pp->wpattern) == 0) {
                                                /* matched */
                                                break;
                                        }
                                }
                        } else {
                                for (pp = patterns; pp; pp = pp->next) {
                                        if (wcswcs(outline, pp->wpattern)
                                            != NULL) {
                                                /* matched */
                                                break;
                                        }
                                }
                        }
                } else if (Fflag) {
                        /* fgrep in byte-oriented handling */
                        char    *fptr;
                        if (iflag) {
                                fptr = istrdup(ptr);
                        } else {
                                fptr = ptr;
                        }
                        if (xflag) {
                                /* fgrep -x */
                                for (pp = patterns; pp; pp = pp->next) {
                                        if (fptr[0] == pp->pattern[0] &&
                                            strcmp(fptr, pp->pattern) == 0) {
                                                /* matched */
                                                break;
                                        }
                                }
                        } else {
                                for (pp = patterns; pp; pp = pp->next) {
                                        if (strstr(fptr, pp->pattern) != NULL) {
                                                /* matched */
                                                break;
                                        }
                                }
                        }
                } else {
                        /* grep or egrep */
                        for (pp = patterns; pp; pp = pp->next) {
                                int     rv;

                                rv = regexec(&pp->re, ptr, 0, NULL, 0);
                                if (rv == REG_OK) {
                                        /* matched */
```

```
                                        break;
                                }

                                switch (rv) {
                                case REG_NOMATCH:
                                        break;
                                case REG_ECHAR:
                                        (void) fprintf(stderr, gettext(
            "%s: input file \"%s\": line %lld: invalid multibyte character\n"),
                                            cmdname, fn, lineno);
                                        break;
                                default:
                                        (void) regerror(rv, &pp->re, errstr,
                                            sizeof (errstr));
                                        (void) fprintf(stderr, gettext(
            "%s: input file \"%s\": line %lld: %s\n"),
                                            cmdname, fn, lineno, errstr);
                                        exit(2);
                                }
                        }
                }

L_next_line:
                /*
                 * Here, if pp points to non-NULL, something has been matched
                 * to the pattern.
                 */
                if (nvflag == (pp != NULL)) {
                        matches++;
                        /*
                         * Handle q, l, and c flags.
                         */
                        if (qflag) {
                                /* no need to continue */
                                /*
                                 * End of this line is ptrend.
                                 * We have read up to ptr + data_len.
                                 */
                                off_t   pos;
                                pos = ptr + data_len - (ptrend + 1);
                                (void) lseek(fd, -pos, SEEK_CUR);
                                exit(0);
                        }
                        if (lflag) {
                                (void) printf("%s\n", fn);
                                break;
                        }
                        if (!cflag) {
                                if (outfn) {
                                        (void) printf("%s:", fn);
                                }
                                if (bflag) {
                                        (void) printf("%lld:", (offset_t)
                                            (line_offset / BSIZE));
                                }
                                if (nflag) {
                                        (void) printf("%lld:", lineno);
                                }
                                *ptrend = '\n';
                                (void) fwrite(ptr, 1, line_len + 1, stdout);
                        }
                        if (ferror(stdout)) {
                                return (0);
                        }
                }
L_skip_line:
```

```
                if (!newlinep)
                        break;

                data_len -= line_len + 1;
                line_offset += line_len + 1;
                ptr = ptrend + 1;
        }

        if (cflag) {
                if (outfn) {
                        (void) printf("%s:", fn);
                }
                if (!qflag) {
                        (void) printf("%lld\n", matches);
                }
        }
        return (matches != 0);
}

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

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

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

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

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

                (void) fprintf(stderr, "\t%s", cmdname);
```

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

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

______***unchanged_portion_omitted_***

```
************************************************************
   13910 Mon Sep 17 14:32:33 2012
new/usr/src/man/man1/grep.1
3047 grep support for -r would be useful
************************************************************
'\" te
.\" Copyright 2012 Nexenta Systems, Inc. All rights reserved.
.\" Copyright 1989 AT&T
.\" Copyright (c) 2008, Sun Microsystems, Inc.  All Rights Reserved
.\" Portions Copyright (c) 1992, X/Open Company Limited  All Rights Reserved
.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
.\" http://www.opengroup.org/bookstore/.
.\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
.\"  This notice shall appear on any product containing this material.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH GREP 1 "Feb 26, 2008"
.SH NAME
grep \- search a file for a pattern
.SH SYNOPSIS
.LP
.nf
\fB/usr/bin/grep\fR [\fB-c\fR | \fB-l\fR |\fB-q\fR] [\fB-r\fR | \fB-R\fR] [\fB-b
    \fIlimited-regular-expression\fR [\fIfilename\fR]...
\fB/usr/bin/grep\fR [\fB-c\fR | \fB-l\fR | \fB-q\fR] [\fB-bhinsvw\fR] \fIlimited
     [\fIfilename\fR]...
.fi

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

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

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

.SH DESCRIPTION
.sp
.LP
The \fBgrep\fR utility searches text files for a pattern and prints all lines
that contain that pattern.  It uses a compact non-deterministic algorithm.
.sp
.LP
Be careful using the characters \fB$\fR, \fB*\fR, \fB[\fR, \fB^\fR, \fB|\fR,
\fB(\fR, \fB)\fR, and \fB\e\fR in the \fIpattern_list\fR because they are also
meaningful to the shell. It is safest to enclose the entire \fIpattern_list\fR
in single quotes \fBa\'\fR\&...\fBa\'\fR\&.
```

```
.sp
.LP
If no files are specified, \fBgrep\fR assumes standard input. Normally, each
line found is copied to standard output. The file name is printed before each
line found if there is more than one input file.
.SS "/usr/bin/grep"
.sp
.LP
The \fB/usr/bin/grep\fR utility uses limited regular expressions like those
described on the \fBregexp\fR(5) manual page to match the patterns.
.SS "/usr/xpg4/bin/grep"
.sp
.LP
The options \fB-E\fR and \fB-F\fR affect the way \fB/usr/xpg4/bin/grep\fR
interprets \fIpattern_list\fR. If \fB-E\fR is specified,
\fB/usr/xpg4/bin/grep\fR interprets \fIpattern_list\fR as a full regular
expression (see \fB-E\fR for description).  If \fB-F\fR is specified,
\fBgrep\fR interprets \fIpattern_list\fR as a fixed string. If neither are
specified, \fBgrep\fR interprets \fIpattern_list\fR as a basic regular
expression as described on \fBregex\fR(5) manual page.
.SH OPTIONS
.sp
.LP
The following options are supported for both \fB/usr/bin/grep\fR and
\fB/usr/xpg4/bin/grep\fR:
.sp
.ne 2
.na
\fB\fB-b\fR\fR
.ad
.RS 6n
Precedes each line by the block number on which it was found. This can be
useful in locating block numbers by context (first block is 0).
.RE

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

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

.sp
.ne 2
.na
\fB\fB-i\fR\fR
.ad
.RS 6n
Ignores upper/lower case distinction during comparisons.
.RE

.sp
.ne 2
.na
```

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

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

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

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

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

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

.sp
.ne 2
.na
\fB\fB-v\fR\fR
.ad
.RS 6n
Prints all lines except those that contain the pattern.
.RE

.sp
```

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

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

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

```
.RE

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

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

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

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

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

.SS "/usr/xpg4/bin/grep"
.sp
.ne 2
.na
\fB\fIpattern\fR\fR
```

```
.ad
.RS 11n
Specifies one or more patterns to be used during the search for input. This
operand is treated as if it were specified as \fB-e\fR \fIpattern_list\fR.
.RE

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

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

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

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

.sp
.LP
or
```

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

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

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

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

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

.SH ENVIRONMENT VARIABLES
.sp
.LP
See \fBenviron\fR(5) for descriptions of the following environment variables
that affect the execution of \fBgrep\fR: \fBLANG\fR, \fBLC_ALL\fR,
\fBLC_COLLATE\fR, \fBLC_CTYPE\fR, \fBLC_MESSAGES\fR, and \fBNLSPATH\fR.
.SH EXIT STATUS
.sp
.LP
The following exit values are returned:
```

```
.sp
.ne 2
.na
\fB\fB0\fR\fR
.ad
.RS 5n
One or more matches were found.
.RE

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

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

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

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

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

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE	ATTRIBUTE VALUE
_
CSI	Enabled
_
Interface Stability	Committed
_
Standard	See \fBstandards\fR(5).
.TE

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