```
**********************************************************
    5556 Thu Nov  1 09:19:11 2012
new/usr/src/head/glob.h
1097 glob(3c) needs to support non-POSIX options
**********************************************************
```

```
/*
 * 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 (c) 1989, 1993
 *      The Regents of the University of California.  All rights reserved.
 *
 * This code is derived from software contributed to Berkeley by
 * Guido van Rossum.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *      @(#)glob.h      8.1 (Berkeley) 6/2/93
 */

/*
 * Copyright 2003 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright (c) 2012 Gary Mills
 */
```



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

#ifndef _GLOB_H
#define _GLOB_H

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

#include <sys/feature_tests.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>

#ifdef __cplusplus
extern "C" {
#endif

struct stat;

typedef struct  glob_t  {
        /* Members required by POSIX */
        size_t  gl_pathc;       /* Total count of paths matched by pattern */
        size_t  gl_pathc;               /* Count of paths matched by pattern */
        char    **gl_pathv;     /* List of matched pathnames */
        size_t  gl_offs;        /* # of slots reserved in gl_pathv */
        /* Non-POSIX extensions, from Openbsd */
        int gl_matchc;          /* Count of paths matching pattern. */
        int gl_flags;           /* Copy of flags parameter to glob. */
        /* Members only accessed when Non-POSIX flags are specified. */
        struct stat **gl_statv; /* Stat entries corresponding to gl_pathv */
        /*
         * Alternate filesystem access methods for glob; replacement
         * versions of closedir(3), readdir(3), opendir(3), stat(2)
         * and lstat(2).
         */
        int (*gl_closedir)(DIR *);
        struct dirent *(*gl_readdir)(DIR *);
        DIR *(*gl_opendir)(const char *);
        int (*gl_lstat)(const char *, struct stat *);
        int (*gl_stat)(const char *, struct stat *);
        /* following are internal to the implementation */
        char    **gl_pathp;             /* gl_pathv + gl_offs */
        int     gl_pathn;               /* # of elements allocated */
}       glob_t;

/*
 * POSIX "flags" argument to glob function.
 * "flags" argument to glob function.
 */
#define GLOB_ERR        0x0001          /* Don't continue on directory error */
#define GLOB_MARK       0x0002          /* Mark directories with trailing / */
#define GLOB_NOSORT     0x0004          /* Don't sort pathnames */
#define GLOB_NOCHECK    0x0008          /* Return unquoted arg if no match */
#define GLOB_DOOFFS     0x0010          /* Ignore gl_offs unless set */
#define GLOB_APPEND     0x0020          /* Append to previous glob_t */
#define GLOB_NOESCAPE   0x0040          /* Backslashes do not quote M-chars */

/*
 * Non-POSIX "flags" argument to glob function, from Openbsd.
 */
#define GLOB_BRACE      0x0080  /* Expand braces ala csh. */
#define GLOB_MAGCHAR    0x0100  /* Pattern had globbing characters. */
#define GLOB_NOMAGIC    0x0200  /* GLOB_NOCHECK without magic chars (csh). */
#define GLOB_QUOTE      0x0400  /* Quote special chars with \. */
```

```
121 #define GLOB_TILDE      0x0800  /* Expand tilde names from the passwd file. */
122 #define GLOB_LIMIT      0x2000  /* Limit pattern match output to ARG_MAX */
123 #define GLOB_KEEPSTAT   0x4000  /* Retain stat data for paths in gl_statv. */
124 #define GLOB_ALTDIRFUNC 0x8000  /* Use alternately specified directory funcs. */

126 /*
127  * Error returns from "glob"
128  */
129 #define GLOB_NOSYS      (-4)            /* function not supported (XPG4) */
130 #define GLOB_NOMATCH    (-3)            /* Pattern does not match */
131 #define GLOB_NOSPACE    (-2)            /* Not enough memory */
132 #define GLOB_ABORTED    (-1)            /* GLOB_ERR set or errfunc return!=0 */
133 #define GLOB_ABEND      GLOB_ABORTED    /* backward compatibility */

135 #if defined(__STDC__)
136 extern int glob(const char *_RESTRICT_KYWD, int, int(*)(const char *, int),
137                 glob_t *_RESTRICT_KYWD);
138 extern void globfree(glob_t *);
139 #else
140 extern int glob();
141 extern void globfree();
142 #endif

144 #ifdef  __cplusplus
145 }
```

*unchanged_portion_omitted_*

****
654 Thu Nov 1 09:19:12 2012
new/usr/src/lib/libc/port/regex/charclass.h
1097 glob(3c) needs to support non-POSIX options
****

```
/*
 * Public domain, 2008, Todd C. Miller <Todd.Miller@courtesan.com>
 *
 * $OpenBSD: charclass.h,v 1.1 2008/10/01 23:04:13 millert Exp $
 */

/*
 * POSIX character class support for fnmatch() and glob().
 */
static struct cclass {
        const char *name;
        int (*isctype)(int);
} cclasses[] = {
        { "alnum",      isalnum },
        { "alpha",      isalpha },
        { "blank",      isblank },
        { "cntrl",      iscntrl },
        { "digit",      isdigit },
        { "graph",      isgraph },
        { "lower",      islower },
        { "print",      isprint },
        { "punct",      ispunct },
        { "space",      isspace },
        { "upper",      isupper },
        { "xdigit",     isxdigit },
        { NULL,         NULL }
};

#define NCCLASSES       (sizeof (cclasses) / sizeof (cclasses[0]) - 1)
```

```
**********************************************************
   26967 Thu Nov  1 09:19:12 2012
new/usr/src/lib/libc/port/regex/glob.c
1097 glob(3c) needs to support non-POSIX options
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (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 (c) 2012 Gary Mills
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      $OpenBSD: glob.c,v 1.39 2012/01/20 07:09:42 tedu Exp $ */
/*
 * Copyright (c) 1989, 1993
 *      The Regents of the University of California.  All rights reserved.
 * This code is MKS code ported to Solaris originally with minimum
 * modifications so that upgrades from MKS would readily integrate.
 * The MKS basis for this modification was:
 *
 * This code is derived from software contributed to Berkeley by
 * Guido van Rossum.
 *      $Id: glob.c 1.31 1994/04/07 22:50:43 mark
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
```



```
 * SUCH DAMAGE.
 * Additional modifications have been made to this code to make it
 * 64-bit clean.
 */

/*
 * glob(3) -- a superset of the one defined in POSIX 1003.2.
 * glob, globfree -- POSIX.2 compatible file name expansion routines.
 *
 * The [!...] convention to negate a range is supported (SysV, Posix, ksh).
 * Copyright 1985, 1991 by Mortice Kern Systems Inc.  All rights reserved.
 *
 * Optional extra services, controlled by flags not defined by POSIX:
 *
 * GLOB_QUOTE:
 *      Escaping convention: \ inhibits any special meaning the following
 *      character might have (except \ at end of string is retained).
 * GLOB_MAGCHAR:
 *      Set in gl_flags if pattern contained a globbing character.
 * GLOB_NOMAGIC:
 *      Same as GLOB_NOCHECK, but it will only append pattern if it did
 *      not contain any magic characters.  [Used in csh style globbing]
 * GLOB_ALTDIRFUNC:
 *      Use alternately specified directory access functions.
 * GLOB_TILDE:
 *      expand ~user/foo to the /home/dir/of/user/foo
 * GLOB_BRACE:
 *      expand {1,2}{a,b} to 1a 1b 2a 2b
 * gl_matchc:
 *      Number of matches in the current invocation of glob.
 * Written by Eric Gisin.
 */

#include <sys/param.h>
#include <sys/stat.h>
#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <ctype.h>
#include <dirent.h>
#include <errno.h>
#include <glob.h>
#include <limits.h>
#include <pwd.h>
#pragma weak _glob = glob
#pragma weak _globfree = globfree

#include "lint.h"
#include <stdio.h>
#include <unistd.h>
#include <limits.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>
#include <glob.h>
#include <errno.h>
#include <fnmatch.h>

#include "charclass.h"
#define GLOB__CHECK     0x80    /* stat generated paths */

#define DOLLAR          '$'
#define DOT             '.'
#define EOS             '\0'
#define LBRACKET        '['
```

```
#define NOT             '!'
#define QUESTION        '?'
#define QUOTE           '\\'
#define RANGE           '-'
#define RBRACKET        ']'
#define SEP             '/'
#define STAR            '*'
#define TILDE           '~'
#define UNDERSCORE      '_'
#define LBRACE          '{'
#define RBRACE          '}'
#define SLASH           '/'
#define COMMA           ','
#define INITIAL 8               /* initial pathv allocation */
#define NULLCPP ((char **)0)    /* Null char ** */
#define NAME_MAX        1024    /* something large */
```

```
#ifndef DEBUG
static int      globit(size_t, const char *, glob_t *, int,
        int (*)(const char *, int), char **);
static int      pstrcmp(const void *, const void *);
static int      append(glob_t *, const char *);
```

```
#define M_QUOTE         0x8000
#define M_PROTECT       0x4000
#define M_MASK          0xffff
#define M_ASCII         0x00ff

typedef ushort_t Char;

#else

#define M_QUOTE         0x80
#define M_PROTECT       0x40
#define M_MASK          0xff
#define M_ASCII         0x7f

typedef char Char;

#endif


#define CHAR(c)         ((Char)((c)&M_ASCII))
#define META(c)         ((Char)((c)|M_QUOTE))
#define M_ALL           META('*')
#define M_END           META(']')
#define M_NOT           META('!')
#define M_ONE           META('?')
#define M_RNG           META('-')
#define M_SET           META('[')
#define M_CLASS         META(':')
#define ismeta(c)       (((c)&M_QUOTE) != 0)

#define GLOB_LIMIT_MALLOC       65536
#define GLOB_LIMIT_STAT         2048
#define GLOB_LIMIT_READDIR      16384

/* Limit of recursion during matching attempts. */
#define GLOB_LIMIT_RECUR        64

struct glob_lim {
        size_t  glim_malloc;
        size_t  glim_stat;
        size_t  glim_readdir;
};
```



```
struct glob_path_stat {
        char            *gps_path;
        struct stat     *gps_stat;
};

static int       compare(const void *, const void *);
static int       compare_gps(const void *, const void *);
static int       g_Ctoc(const Char *, char *, uint_t);
static int       g_lstat(Char *, struct stat *, glob_t *);
static DIR      *g_opendir(Char *, glob_t *);
static Char     *g_strchr(const Char *, int);
static int       g_strncmp(const Char *, const char *, size_t);
static int       g_stat(Char *, struct stat *, glob_t *);
static int       glob0(const Char *, glob_t *, struct glob_lim *,
                       int (*)(const char *, int));
static int       glob1(Char *, Char *, glob_t *, struct glob_lim *,
                       int (*)(const char *, int));
static int       glob2(Char *, Char *, Char *, Char *, Char *, Char *,
                       glob_t *, struct glob_lim *,
                       int (*)(const char *, int));
static int       glob3(Char *, Char *, Char *, Char *, Char *,
                       Char *, Char *, glob_t *, struct glob_lim *,
                       int (*)(const char *, int));
static int       globextend(const Char *, glob_t *, struct glob_lim *,
                       struct stat *);
static
const Char      *globtilde(const Char *, Char *, size_t, glob_t *);
static int       globexp1(const Char *, glob_t *, struct glob_lim *,
                       int (*)(const char *, int));
static int       globexp2(const Char *, const Char *, glob_t *,
                       struct glob_lim *, int (*)(const char *, int));
static int       match(Char *, Char *, Char *, int);
#ifdef DEBUG
static void      qprintf(const char *, Char *);
#endif

int
glob(const char *pattern, int flags, int (*errfunc)(const char *, int),
    glob_t *pglob)
{
        const uchar_t *patnext;
        int c;
        Char *bufnext, *bufend, patbuf[MAXPATHLEN];
        struct glob_lim limit = { 0, 0, 0 };

        if (strnlen(pattern, PATH_MAX) == PATH_MAX)
                return (GLOB_NOMATCH);

        patnext = (uchar_t *)pattern;
        if (!(flags & GLOB_APPEND)) {
                pglob->gl_pathc = 0;
                pglob->gl_pathv = NULL;
                if ((flags & GLOB_KEEPSTAT) != 0)
                        pglob->gl_statv = NULL;
                if (!(flags & GLOB_DOOFFS))
                        pglob->gl_offs = 0;
        }
        pglob->gl_flags = flags & ~GLOB_MAGCHAR;
        pglob->gl_matchc = 0;

        if (pglob->gl_offs < 0 || pglob->gl_pathc < 0 ||
            pglob->gl_offs >= INT_MAX || pglob->gl_pathc >= INT_MAX ||
            pglob->gl_pathc >= INT_MAX - pglob->gl_offs - 1)
                return (GLOB_NOSPACE);

        bufnext = patbuf;
```

```
229         bufend = bufnext + MAXPATHLEN - 1;
230         if (flags & GLOB_NOESCAPE)
231                 while (bufnext < bufend && (c = *patnext++) != EOS)
232                         *bufnext++ = c;
233         else {
234                 /* Protect the quoted characters. */
235                 while (bufnext < bufend && (c = *patnext++) != EOS)
236                         if (c == QUOTE) {
237                                 if ((c = *patnext++) == EOS) {
238                                         c = QUOTE;
239                                         --patnext;
240                                 }
241                                 *bufnext++ = c | M_PROTECT;
242                         } else
243                                 *bufnext++ = c;
244         }
245         *bufnext = EOS;

247         if (flags & GLOB_BRACE)
248                 return (globexp1(patbuf, pglob, &limit, errfunc));
249         else
250                 return (glob0(patbuf, pglob, &limit, errfunc));
251 }

253 /*
254  * Expand recursively a glob {} pattern. When there is no more expansion
255  * invoke the standard globbing routine to glob the rest of the magic
256  * characters
 75  * Free all space consumed by glob.
257  */
258 static int
259 globexp1(const Char *pattern, glob_t *pglob, struct glob_lim *limitp,
260     int (*errfunc)(const char *, int))
 77 void
 78 globfree(glob_t *gp)
261 {
262         const Char* ptr = pattern;
 80         size_t i;

264         /* Protect a single {}, for find(1), like csh */
265         if (pattern[0] == LBRACE && pattern[1] == RBRACE && pattern[2] == EOS)
266                 return (glob0(pattern, pglob, limitp, errfunc));
 82         if (gp->gl_pathv == 0)
 83                 return;

268         if ((ptr = (const Char *) g_strchr(ptr, LBRACE)) != NULL)
269                 return (globexp2(ptr, pattern, pglob, limitp, errfunc));
 85         for (i = gp->gl_offs; i < gp->gl_offs + gp->gl_pathc; ++i)
 86                 free(gp->gl_pathv[i]);
 87         free((void *)gp->gl_pathv);

271         return (glob0(pattern, pglob, limitp, errfunc));
 89         gp->gl_pathc = 0;
 90         gp->gl_pathv = NULLCPP;
272 }


275 /*
276  * Recursive brace globbing helper. Tries to expand a single brace.
277  * If it succeeds then it invokes globexp1 with the new pattern.
278  * If it fails then it tries to glob the rest of the pattern and returns.
 94  * Do filename expansion.
279  */
280 static int
281 globexp2(const Char *ptr, const Char *pattern, glob_t *pglob,
282     struct glob_lim *limitp, int (*errfunc)(const char *, int))
```



```
 96 int
 97 glob(const char *pattern, int flags,
 98         int (*errfn)(const char *, int), glob_t *gp)
283 {
284         int     i, rv;
285         Char   *lm, *ls;
286         const Char *pe, *pm, *pl;
287         Char    patbuf[MAXPATHLEN];
100         int rv;
101         size_t i;
102         size_t ipathc;
103         char    *path;

289         /* copy part up to the brace */
290         for (lm = patbuf, pm = pattern; pm != ptr; *lm++ = *pm++)
291                 ;
292         *lm = EOS;
293         ls = lm;
105         if ((flags & GLOB_DOOFFS) == 0)
106                 gp->gl_offs = 0;

295         /* Find the balanced brace */
296         for (i = 0, pe = ++ptr; *pe; pe++)
297                 if (*pe == LBRACKET) {
298                         /* Ignore everything between [] */
299                         for (pm = pe++; *pe != RBRACKET && *pe != EOS; pe++)
300                                 ;
301                         if (*pe == EOS) {
302                                 /*
303                                  * We could not find a matching RBRACKET.
304                                  * Ignore and just look for RBRACE
305                                  */
306                                 pe = pm;
307                         }
308                 } else if (*pe == LBRACE)
309                         i++;
310                 else if (*pe == RBRACE) {
311                         if (i == 0)
312                                 break;
313                         i--;
314                 }
108         if (!(flags & GLOB_APPEND)) {
109                 gp->gl_pathc = 0;
110                 gp->gl_pathn = gp->gl_offs + INITIAL;
111                 gp->gl_pathv = (char **)malloc(sizeof (char *) * gp->gl_pathn);

316         /* Non matching braces; just glob the pattern */
317         if (i != 0 || *pe == EOS)
318                 return (glob0(patbuf, pglob, limitp, errfunc));
113                 if (gp->gl_pathv == NULLCPP)
114                         return (GLOB_NOSPACE);
115                 gp->gl_pathp = gp->gl_pathv + gp->gl_offs;

320         for (i = 0, pl = pm = ptr; pm <= pe; pm++) {
321                 switch (*pm) {
322                 case LBRACKET:
323                         /* Ignore everything between [] */
324                         for (pl = pm++; *pm != RBRACKET && *pm != EOS; pm++)
325                                 ;
326                         if (*pm == EOS) {
327                                 /*
328                                  * We could not find a matching RBRACKET.
329                                  * Ignore and just look for RBRACE
330                                  */
331                                 pm = pl;
117                 for (i = 0; i < gp->gl_offs; ++i)
```

```
118                         gp->gl_pathv[i] = NULL;
332                         }
333                         break;

335                 case LBRACE:
336                         i++;
337                         break;
121         if ((path = malloc(strlen(pattern)+1)) == NULL)
122                 return (GLOB_NOSPACE);

339                 case RBRACE:
340                         if (i) {
341                                 i--;
342                                 break;
343                         }
344                         /* FALLTHROUGH */
345                 case COMMA:
346                         if (i && *pm == COMMA)
347                                 break;
348                         else {
349                                 /* Append the current string */
350                                 for (lm = ls; (pl < pm); *lm++ = *pl++)
351                                         ;
124         ipathc = gp->gl_pathc;
125         rv = globit(0, pattern, gp, flags, errfn, &path);

127         if (rv == GLOB_ABORTED) {
353                                 /*
354                                  * Append the rest of the pattern after the
355                                  * closing brace
129                  * User's error function returned non-zero, or GLOB_ERR was
130                  * set, and we encountered a directory we couldn't search.
356                                  */
357                                 for (pl = pe + 1; (*lm++ = *pl++) != EOS; )
358                                         ;

360                                 /* Expand the current pattern */
361 #ifdef DEBUG
362                                 qprintf("globexp2:", patbuf);
363 #endif
364                                 rv = globexp1(patbuf, pglob, limitp, errfunc);
365                                 if (rv && rv != GLOB_NOMATCH)
366                                         return (rv);

368                                 /* move after the comma, to the next string */
369                                 pl = pm + 1;
132                 free(path);
133                 return (GLOB_ABORTED);
370                         }
371                         break;

373                 default:
374                         break;
136         i = gp->gl_pathc - ipathc;
137         if (i >= 1 && !(flags & GLOB_NOSORT)) {
138                 qsort((char *)(gp->gl_pathp+ipathc), i, sizeof (char *),
139                     pstrcmp);
375                 }
376         }
377         return (0);
378 }



382 /*
383  * expand tilde from the passwd file.
```



```
384  */
385 static const Char *
386 globtilde(const Char *pattern, Char *patbuf, size_t patbuf_len, glob_t *pglob)
387 {
388         struct passwd *pwd;
389         char *h;
390         const Char *p;
391         Char *b, *eb;

393         if (*pattern != TILDE || !(pglob->gl_flags & GLOB_TILDE))
394                 return (pattern);

396         /* Copy up to the end of the string or / */
397         eb = &patbuf[patbuf_len - 1];
398         for (p = pattern + 1, h = (char *)patbuf;
399             h < (char *)eb && *p && *p != SLASH; *h++ = *p++)
400                 ;

402         *h = EOS;

404 #if 0
405         if (h == (char *)eb)
406                 return (what);
407 #endif

409         if (((char *)patbuf)[0] == EOS) {
410                 /*
411                  * handle a plain ~ or ~/ by expanding $HOME
412                  * first and then trying the password file
413                  */
414                 if (issetugid() != 0 || (h = getenv("HOME")) == NULL) {
415                         if ((pwd = getpwuid(getuid())) == NULL)
416                                 return (pattern);
141         if (i == 0) {
142                 if (flags & GLOB_NOCHECK)
143                         (void) append(gp, pattern);
417                         else
418                                 h = pwd->pw_dir;
145                         rv = GLOB_NOMATCH;
419                 }
420         } else {
421                 /*
422                  * Expand a ~user
423                  */
424                 if ((pwd = getpwnam((char *)patbuf)) == NULL)
425                         return (pattern);
426                 else
427                         h = pwd->pw_dir;
428         }
147         gp->gl_pathp[gp->gl_pathc] = NULL;
148         free(path);

430         /* Copy the home directory */
431         for (b = patbuf; b < eb && *h; *b++ = *h++)
432                 ;

434         /* Append the rest of the pattern */
435         while (b < eb && (*b++ = *p++) != EOS)
436                 ;
437         *b = EOS;

439         return (patbuf);
440 }

442 static int
443 g_strncmp(const Char *s1, const char *s2, size_t n)
```

```
444 {
445         int rv = 0;

447         while (n--) {
448                 rv = *(Char *)s1 - *(const unsigned char *)s2++;
449                 if (rv)
450                         break;
451                 if (*s1++ == '\0')
452                         break;
453         }
454         return (rv);
455 }

457 static int
458 g_charclass(const Char **patternp, Char **bufnextp)
459 {
460         const Char *pattern = *patternp + 1;
461         Char *bufnext = *bufnextp;
462         const Char *colon;
463         struct cclass *cc;
464         size_t len;

466         if ((colon = g_strchr(pattern, ':')) == NULL || colon[1] != ']')
467                 return (1);     /* not a character class */

469         len = (size_t)(colon - pattern);
470         for (cc = cclasses; cc->name != NULL; cc++) {
471                 if (!g_strncmp(pattern, cc->name, len) && cc->name[len] == '\0')
472                         break;
473         }
474         if (cc->name == NULL)
475                 return (-1);    /* invalid character class */
476         *bufnext++ = M_CLASS;
477         *bufnext++ = (Char)(cc - &cclasses[0]);
478         *bufnextp = bufnext;
479         *patternp += len + 3;

481         return (0);
482 }

484 /*
485  * The main glob() routine: compiles the pattern (optionally processing
486  * quotes), calls glob1() to do the real pattern matching, and finally
487  * sorts the list (unless unsorted operation is requested).  Returns 0
488  * if things went well, nonzero if errors occurred.  It is not an error
489  * to find no matches.
155  * Recursive routine to match glob pattern, and walk directories.
490  */
491 static int
492 glob0(const Char *pattern, glob_t *pglob, struct glob_lim *limitp,
493     int (*errfunc)(const char *, int))
157 int
158 globit(size_t dend, const char *sp, glob_t *gp, int flags,
159         int (*errfn)(const char *, int), char **path)
494 {
495         const Char *qpatnext;
496         int c, err, oldpathc;
497         Char *bufnext, patbuf[MAXPATHLEN];
161         size_t n;
162         size_t m;
163         ssize_t end = 0;        /* end of expanded directory */
164         char *pat = (char *)sp; /* pattern component */
165         char *dp = (*path) + dend;
166         int expand = 0;         /* path has pattern */
167         char *cp;
168         struct stat64 sb;
```



```
169         DIR *dirp;
170         struct dirent64 *d;
171         int err;

499         qpatnext = globtilde(pattern, patbuf, MAXPATHLEN, pglob);
500         oldpathc = pglob->gl_pathc;
501         bufnext = patbuf;

503         /* We don't need to check for buffer overflow any more. */
504         while ((c = *qpatnext++) != EOS) {
505                 switch (c) {
506                 case LBRACKET:
507                         c = *qpatnext;
508                         if (c == NOT)
509                                 ++qpatnext;
510                         if (*qpatnext == EOS ||
511                             g_strchr(qpatnext+1, RBRACKET) == NULL) {
512                                 *bufnext++ = LBRACKET;
513                                 if (c == NOT)
514                                         --qpatnext;
515                                 break;
173         for (;;)
174                 switch (*dp++ = *(unsigned char *)sp++) {
175                 case '\0':      /* end of source path */
176                         if (expand)
177                                 goto Expand;
178                         else {
179                                 if (!(flags & GLOB_NOCHECK) ||
180                                     flags & (GLOB__CHECK|GLOB_MARK))
181                                         if (stat64(*path, &sb) < 0) {
182                                                 return (0);
516                         }
517                         *bufnext++ = M_SET;
518                         if (c == NOT)
519                                 *bufnext++ = M_NOT;
520                         c = *qpatnext++;
521                         do {
522                                 if (c == LBRACKET && *qpatnext == ':') {
523                                         do {
524                                                 err = g_charclass(&qpatnext,
525                                                     &bufnext);
526                                                 if (err)
527                                                         break;
528                                                 c = *qpatnext++;
529                                         } while (c == LBRACKET &&
530                                             *qpatnext == ':');
531                                         if (err == -1 &&
532                                             !(pglob->gl_flags & GLOB_NOCHECK))
533                                                 return (GLOB_NOMATCH);
534                                         if (c == RBRACKET)
535                                                 break;
184                                 if (flags & GLOB_MARK && S_ISDIR(sb.st_mode)) {
185                                         *dp = '\0';
186                                         *--dp = '/';
536                                 }
537                                 *bufnext++ = CHAR(c);
538                                 if (*qpatnext == RANGE &&
539                                     (c = qpatnext[1]) != RBRACKET) {
540                                         *bufnext++ = M_RNG;
541                                         *bufnext++ = CHAR(c);
542                                         qpatnext += 2;
543                                 }
544                         } while ((c = *qpatnext++) != RBRACKET);
545                         pglob->gl_flags |= GLOB_MAGCHAR;
546                         *bufnext++ = M_END;
547                         break;
```

```
548                 case QUESTION:
549                         pglob->gl_flags |= GLOB_MAGCHAR;
550                         *bufnext++ = M_ONE;
551                         break;
552                 case STAR:
553                         pglob->gl_flags |= GLOB_MAGCHAR;
554                         /*
555                          * collapse adjacent stars to one,
556                          * to avoid exponential behavior
557                          */
558                         if (bufnext == patbuf || bufnext[-1] != M_ALL)
559                                 *bufnext++ = M_ALL;
560                         break;
561                 default:
562                         *bufnext++ = CHAR(c);
563                         break;
564                 }
565         }
566         *bufnext = EOS;
567 #ifdef DEBUG
568         qprintf("glob0:", patbuf);
569 #endif

571         if ((err = glob1(patbuf, patbuf+MAXPATHLEN-1, pglob, limitp, errfunc))
572             != 0)
573                 return (err);

575         /*
576          * If there was no match we are going to append the pattern
577          * if GLOB_NOCHECK was specified or if GLOB_NOMAGIC was specified
578          * and the pattern did not contain any magic characters
579          * GLOB_NOMAGIC is there just for compatibility with csh.
580          */
581         if (pglob->gl_pathc == oldpathc) {
582                 if ((pglob->gl_flags & GLOB_NOCHECK) ||
583                     ((pglob->gl_flags & GLOB_NOMAGIC) &&
584                     !(pglob->gl_flags & GLOB_MAGCHAR)))
585                         return (globextend(pattern, pglob, limitp, NULL));
586                 else
587                         return (GLOB_NOMATCH);
588         }
589         if (!(pglob->gl_flags & GLOB_NOSORT)) {
590                 if ((pglob->gl_flags & GLOB_KEEPSTAT)) {
591                         /* Keep the paths and stat info synced during sort */
592                         struct glob_path_stat *path_stat;
593                         int i;
594                         int n = pglob->gl_pathc - oldpathc;
595                         int o = pglob->gl_offs + oldpathc;

597                         if ((path_stat = calloc(n, sizeof (*path_stat))) ==
598                             NULL)
188                                 if (append(gp, *path) < 0) {
599                                 return (GLOB_NOSPACE);
600                         for (i = 0; i < n; i++) {
601                                 path_stat[i].gps_path = pglob->gl_pathv[o + i];
602                                 path_stat[i].gps_stat = pglob->gl_statv[o + i];
603                         }
604                         qsort(path_stat, n, sizeof (*path_stat), compare_gps);
605                         for (i = 0; i < n; i++) {
606                                 pglob->gl_pathv[o + i] = path_stat[i].gps_path;
607                                 pglob->gl_statv[o + i] = path_stat[i].gps_stat;
608                         }
609                         free(path_stat);
610                 } else {
611                         qsort(pglob->gl_pathv + pglob->gl_offs + oldpathc,
612                             pglob->gl_pathc - oldpathc, sizeof (char *),
```



```
613                             compare);
614                 }
615         }
616         return (0);
617 }

619 static int
620 compare(const void *p, const void *q)
621 {
622         return (strcmp(*(char **)p, *(char **)q));
623 }

625 static int
626 compare_gps(const void *_p, const void *_q)
627 {
628         const struct glob_path_stat *p = (const struct glob_path_stat *)_p;
629         const struct glob_path_stat *q = (const struct glob_path_stat *)_q;

631         return (strcmp(p->gps_path, q->gps_path));
632 }

634 static int
635 glob1(Char *pattern, Char *pattern_last, glob_t *pglob,
636     struct glob_lim *limitp, int (*errfunc)(const char *, int))
637 {
638         Char pathbuf[MAXPATHLEN];

640         /* A null pathname is invalid -- POSIX 1003.1 sect. 2.4. */
641         if (*pattern == EOS)
642                 return (0);
643         return (glob2(pathbuf, pathbuf+MAXPATHLEN-1,
644             pathbuf, pathbuf+MAXPATHLEN-1,
645             pattern, pattern_last, pglob, limitp, errfunc));
646 }

648 /*
649  * The functions glob2 and glob3 are mutually recursive; there is one level
650  * of recursion for each segment in the pattern that contains one or more
651  * meta characters.
652  */
653 static int
654 glob2(Char *pathbuf, Char *pathbuf_last, Char *pathend, Char *pathend_last,
655     Char *pattern, Char *pattern_last, glob_t *pglob,
656     struct glob_lim *limitp, int (*errfunc)(const char *, int))
657 {
658         struct stat sb;
659         Char *p, *q;
660         int anymeta;

662         /*
663          * Loop over pattern segments until end of pattern or until
664          * segment with meta character found.
665          */
666         for (anymeta = 0; ; ) {
667                 if (*pattern == EOS) {          /* End of pattern? */
668                         *pathend = EOS;

670                         if ((pglob->gl_flags & GLOB_LIMIT) &&
671                             limitp->glim_stat++ >= GLOB_LIMIT_STAT) {
672                                 errno = 0;
673                                 *pathend++ = SEP;
674                                 *pathend = EOS;
675                                 return (GLOB_NOSPACE);
676                         }
677                         if (g_lstat(pathbuf, &sb, pglob))
678                                 return (0);
```

```
193                         /*NOTREACHED*/

680                         if (((pglob->gl_flags & GLOB_MARK) &&
681                             pathend[-1] != SEP) && (S_ISDIR(sb.st_mode) ||
682                             (S_ISLNK(sb.st_mode) &&
683                             (g_stat(pathbuf, &sb, pglob) == 0) &&
684                             S_ISDIR(sb.st_mode)))) {
685                                 if (pathend+1 > pathend_last)
686                                         return (1);
687                                 *pathend++ = SEP;
688                                 *pathend = EOS;
689                         }
690                         ++pglob->gl_matchc;
691                         return (globextend(pathbuf, pglob, limitp, &sb));
692                 }
195                 case '*':
196                 case '?':
197                 case '[':
198                 case '\\':
199                         ++expand;
200                         break;

694                 /* Find end of next segment, copy tentatively to pathend. */
695                 q = pathend;
696                 p = pattern;
697                 while (*p != EOS && *p != SEP) {
698                         if (ismeta(*p))
699                                 anymeta = 1;
700                         if (q+1 > pathend_last)
701                                 return (1);
702                         *q++ = *p++;
703                 }
202                 case '/':
203                         if (expand)
204                                 goto Expand;
205                         end = dp - *path;
206                         pat = (char *)sp;
207                         break;

705                 if (!anymeta) {         /* No expansion, do next segment. */
706                         pathend = q;
707                         pattern = p;
708                         while (*pattern == SEP) {
709                                 if (pathend+1 > pathend_last)
710                                         return (1);
711                                 *pathend++ = *pattern++;
712                         }
713                 } else
714                         /* Need expansion, recurse. */
715                         return (glob3(pathbuf, pathbuf_last, pathend,
716                             pathend_last, pattern, p, pattern_last,
717                             pglob, limitp, errfunc));
718         }
719         /* NOTREACHED */
720 }

722 static int
723 glob3(Char *pathbuf, Char *pathbuf_last, Char *pathend, Char *pathend_last,
724     Char *pattern, Char *restpattern, Char *restpattern_last, glob_t *pglob,
725     struct glob_lim *limitp, int (*errfunc)(const char *, int))
726 {
727         struct dirent *dp;
728         DIR *dirp;
729         int err;
730         char buf[MAXPATHLEN];
731         struct dirent *(*readdirfunc)(DIR *);
```



```
733         if (pathend > pathend_last)
734                 return (1);
735         *pathend = EOS;
736         errno = 0;

738         if ((dirp = g_opendir(pathbuf, pglob)) == NULL) {
739                 /* TODO: don't call for ENOENT or ENOTDIR? */
740                 if (errfunc) {
741                         if (g_Ctoc(pathbuf, buf, sizeof (buf)))
209                 Expand:
210                         /* determine directory and open it */
211                         (*path)[end] = '\0';
212                         dirp = opendir(**path == '\0' ? "." : *path);
213                         if (dirp == NULL) {
214                                 if (errfn != 0 && errfn(*path, errno) != 0 ||
215                                     flags&GLOB_ERR) {
742                                         return (GLOB_ABORTED);
743                                 if (errfunc(buf, errno) ||
744                                     pglob->gl_flags & GLOB_ERR)
745                                         return (GLOB_ABORTED);
746                 }
747                 return (0);
748         }

750         err = 0;

752         /* Search directory for matching names. */
753         if (pglob->gl_flags & GLOB_ALTDIRFUNC)
754                 readdirfunc = pglob->gl_readdir;
755         else
756                 readdirfunc = readdir;
757         while ((dp = (*readdirfunc)(dirp))) {
758                 uchar_t *sc;
759                 Char *dc;

761                 if ((pglob->gl_flags & GLOB_LIMIT) &&
762                     limitp->glim_readdir++ >= GLOB_LIMIT_READDIR) {
763                         errno = 0;
764                         *pathend++ = SEP;
765                         *pathend = EOS;
766                         err = GLOB_NOSPACE;
767                         break;
221                         /* extract pattern component */
222                         n = sp - pat;
223                         if ((cp = malloc(n)) == NULL) {
224                                 (void) closedir(dirp);
225                                 return (GLOB_NOSPACE);
768                 }
227                         pat = memcpy(cp, pat, n);
228                         pat[n-1] = '\0';
229                         if (*--sp != '\0')
230                                 flags |= GLOB__CHECK;

770                 /* Initial DOT must be matched literally. */
771                 if (dp->d_name[0] == DOT && *pattern != DOT)
772                         continue;
773                 dc = pathend;
774                 sc = (uchar_t *)dp->d_name;
775                 while (dc < pathend_last && (*dc++ = *sc++) != EOS)
776                         ;
777                 if (dc >= pathend_last) {
778                         *dc = EOS;
779                         err = 1;
780                         break;
232                         /* expand path to max. expansion */
```

```
233                         n = dp - *path;
234                         *path = realloc(*path,
235                             strlen(*path) + NAME_MAX + strlen(sp) + 1);
236                         if (*path == NULL) {
237                                 (void) closedir(dirp);
238                                 free(pat);
239                                 return (GLOB_NOSPACE);
781                 }
241                         dp = (*path) + n;

783                 if (!match(pathend, pattern, restpattern, GLOB_LIMIT_RECUR)) {
784                         *pathend = EOS;
243                         /* read directory and match entries */
244                         err = 0;
245                         while ((d = readdir64(dirp)) != NULL) {
246                                 cp = d->d_name;
247                                 if ((flags&GLOB_NOESCAPE)
248                                     ? fnmatch(pat, cp, FNM_PERIOD|FNM_NOESCAPE)
249                                     : fnmatch(pat, cp, FNM_PERIOD))
785                                 continue;
786                 }
787                 err = glob2(pathbuf, pathbuf_last, --dc, pathend_last,
788                     restpattern, restpattern_last, pglob, limitp,
789                     errfunc);
790                 if (err)

252                                 n = strlen(cp);
253                                 (void) memcpy((*path) + end, cp, n);
254                                 m = dp - *path;
255                                 err = globit(end+n, sp, gp, flags, errfn, path);
256                                 dp = (*path) + m;   /* globit can move path */
257                                 if (err != 0)
791                         break;
792         }

794         if (pglob->gl_flags & GLOB_ALTDIRFUNC)
795                 (*pglob->gl_closedir)(dirp);
796         else
797                 closedir(dirp);
261                         (void) closedir(dirp);
262                         free(pat);
798         return (err);
264                 }
265                 /* NOTREACHED */
799 }


802 /*
803  * Extend the gl_pathv member of a glob_t structure to accommodate a new item,
804  * add the new item, and update gl_pathc.
805  *
806  * This assumes the BSD realloc, which only copies the block when its size
807  * crosses a power-of-two boundary; for v7 realloc, this would cause quadratic
808  * behavior.
809  *
810  * Return 0 if new item added, error code if memory couldn't be allocated.
811  *
812  * Invariant of the glob_t structure:
813  *      Either gl_pathc is zero and gl_pathv is NULL; or gl_pathc > 0 and
814  *      gl_pathv points to (gl_offs + gl_pathc + 1) items.
269  * Comparison routine for two name arguments, called by qsort.
815  */
816 static int
817 globextend(const Char *path, glob_t *pglob, struct glob_lim *limitp,
818     struct stat *sb)
271 int
```



```
272 pstrcmp(const void *npp1, const void *npp2)
819 {
820         char **pathv;
821         ssize_t i;
822         size_t newn, len;
823         char *copy = NULL;
824         const Char *p;
825         struct stat **statv;

827         newn = 2 + pglob->gl_pathc + pglob->gl_offs;
828         if (pglob->gl_offs >= INT_MAX ||
829             pglob->gl_pathc >= INT_MAX ||
830             newn >= INT_MAX ||
831             SIZE_MAX / sizeof (*pathv) <= newn ||
832             SIZE_MAX / sizeof (*statv) <= newn) {
833         nospace:
834                 for (i = pglob->gl_offs; i < (ssize_t)(newn - 2); i++) {
835                         if (pglob->gl_pathv && pglob->gl_pathv[i])
836                                 free(pglob->gl_pathv[i]);
837                         if ((pglob->gl_flags & GLOB_KEEPSTAT) != 0 &&
838                             pglob->gl_pathv && pglob->gl_pathv[i])
839                                 free(pglob->gl_statv[i]);
840                 }
841                 if (pglob->gl_pathv) {
842                         free(pglob->gl_pathv);
843                         pglob->gl_pathv = NULL;
844                 }
845                 if ((pglob->gl_flags & GLOB_KEEPSTAT) != 0 &&
846                     pglob->gl_statv) {
847                         free(pglob->gl_statv);
848                         pglob->gl_statv = NULL;
849                 }
850                 return (GLOB_NOSPACE);
851         }

853         pathv = realloc(pglob->gl_pathv, newn * sizeof (*pathv));
854         if (pathv == NULL)
855                 goto nospace;
856         if (pglob->gl_pathv == NULL && pglob->gl_offs > 0) {
857                 /* first time around -- clear initial gl_offs items */
858                 pathv += pglob->gl_offs;
859                 for (i = pglob->gl_offs; --i >= 0; )
860                         *--pathv = NULL;
861         }
862         pglob->gl_pathv = pathv;

864         if ((pglob->gl_flags & GLOB_KEEPSTAT) != 0) {
865                 statv = realloc(pglob->gl_statv, newn * sizeof (*statv));
866                 if (statv == NULL)
867                         goto nospace;
868                 if (pglob->gl_statv == NULL && pglob->gl_offs > 0) {
869                         /* first time around -- clear initial gl_offs items */
870                         statv += pglob->gl_offs;
871                         for (i = pglob->gl_offs; --i >= 0; )
872                                 *--statv = NULL;
873                 }
874                 pglob->gl_statv = statv;
875                 if (sb == NULL)
876                         statv[pglob->gl_offs + pglob->gl_pathc] = NULL;
877                 else {
878                         limitp->glim_malloc += sizeof (**statv);
879                         if ((pglob->gl_flags & GLOB_LIMIT) &&
880                             limitp->glim_malloc >= GLOB_LIMIT_MALLOC) {
881                                 errno = 0;
882                                 return (GLOB_NOSPACE);
883                         }
```

```
884                         if ((statv[pglob->gl_offs + pglob->gl_pathc] =
885                             malloc(sizeof (**statv))) == NULL)
886                                 goto copy_error;
887                         memcpy(statv[pglob->gl_offs + pglob->gl_pathc], sb,
888                             sizeof (*sb));
889                 }
890                 statv[pglob->gl_offs + pglob->gl_pathc + 1] = NULL;
891         }

893         for (p = path; *p++; )
894                 ;
895         len = (size_t)(p - path);
896         limitp->glim_malloc += len;
897         if ((copy = malloc(len)) != NULL) {
898                 if (g_Ctoc(path, copy, len)) {
899                         free(copy);
900                         return (GLOB_NOSPACE);
901                 }
902                 pathv[pglob->gl_offs + pglob->gl_pathc++] = copy;
903         }
904         pathv[pglob->gl_offs + pglob->gl_pathc] = NULL;

906         if ((pglob->gl_flags & GLOB_LIMIT) &&
907             (newn * sizeof (*pathv)) + limitp->glim_malloc >
908             GLOB_LIMIT_MALLOC) {
909                 errno = 0;
910                 return (GLOB_NOSPACE);
911         }
912         copy_error:
913         return (copy == NULL ? GLOB_NOSPACE : 0);
274         return (strcoll(*(char **)npp1, *(char **)npp2));
914 }


917 /*
918  * pattern matching function for filenames.  Each occurrence of the *
919  * pattern causes a recursion level.
278  * Add a new matched filename to the glob_t structure, increasing the
279  * size of that array, as required.
920  */
921 static int
922 match(Char *name, Char *pat, Char *patend, int recur)
281 int
282 append(glob_t *gp, const char *str)
923 {
924         int ok, negate_range;
925         Char c, k;
284         char *cp;

927         if (recur-- == 0)
286         if ((cp = malloc(strlen(str)+1)) == NULL)
928                 return (GLOB_NOSPACE);
288         gp->gl_pathp[gp->gl_pathc++] = strcpy(cp, str);

930         while (pat < patend) {
931                 c = *pat++;
932                 switch (c & M_MASK) {
933                 case M_ALL:
934                         while (pat < patend && (*pat & M_MASK) == M_ALL)
935                                 pat++;  /* eat consecutive '*' */
936                         if (pat == patend)
937                                 return (1);
938                         do {
939                                 if (match(name, pat, patend, recur))
940                                         return (1);
941                         } while (*name++ != EOS);
```



```
942                         return (0);
943                 case M_ONE:
944                         if (*name++ == EOS)
945                                 return (0);
946                         break;
947                 case M_SET:
948                         ok = 0;
949                         if ((k = *name++) == EOS)
950                                 return (0);
951                         if ((negate_range = ((*pat & M_MASK) == M_NOT)) != EOS)
952                                 ++pat;
953                         while (((c = *pat++) & M_MASK) != M_END) {
954                                 if ((c & M_MASK) == M_CLASS) {
955                                         Char idx = *pat & M_MASK;
956                                         if (idx < NCCLASSES &&
957                                             cclasses[idx].isctype(k))
958                                                 ok = 1;
959                                         ++pat;
290         if ((gp->gl_pathc + gp->gl_offs) >= gp->gl_pathn) {
291                 gp->gl_pathn *= 2;
292                 gp->gl_pathv = (char **)realloc((void *)gp->gl_pathv,
293                     gp->gl_pathn * sizeof (char *));
294                 if (gp->gl_pathv == NULLCPP)
295                         return (GLOB_NOSPACE);
296                 gp->gl_pathp = gp->gl_pathv + gp->gl_offs;
960                                 }
961                                 if ((*pat & M_MASK) == M_RNG) {
962                                         if (c <= k && k <= pat[1])
963                                                 ok = 1;
964                                         pat += 2;
965                                 } else if (c == k)
966                                         ok = 1;
967                         }
968                         if (ok == negate_range)
969                                 return (0);
970                         break;
971                 default:
972                         if (*name++ != c)
973                                 return (0);
974                         break;
975                 }
976         }
977         return (*name == EOS);
978 }

980 /* Free allocated data belonging to a glob_t structure. */
981 void
982 globfree(glob_t *pglob)
983 {
984         int i;
985         char **pp;

987         if (pglob->gl_pathv != NULL) {
988                 pp = pglob->gl_pathv + pglob->gl_offs;
989                 for (i = pglob->gl_pathc; i--; ++pp)
990                         if (*pp)
991                                 free(*pp);
992                 free(pglob->gl_pathv);
993                 pglob->gl_pathv = NULL;
994         }
995         if ((pglob->gl_flags & GLOB_KEEPSTAT) != 0 &&
996             pglob->gl_statv != NULL) {
997                 for (i = 0; i < pglob->gl_pathc; i++) {
998                         if (pglob->gl_statv[i] != NULL)
999                                 free(pglob->gl_statv[i]);
1000                }
```

```
1001                 free(pglob->gl_statv);
1002                 pglob->gl_statv = NULL;
1003         }
1004 }

1006 static DIR *
1007 g_opendir(Char *str, glob_t *pglob)
1008 {
1009         char buf[MAXPATHLEN];

1011         if (!*str)
1012                 strlcpy(buf, ".", sizeof (buf));
1013         else {
1014                 if (g_Ctoc(str, buf, sizeof (buf)))
1015                         return (NULL);
1016         }

1018         if (pglob->gl_flags & GLOB_ALTDIRFUNC)
1019                 return ((*pglob->gl_opendir)(buf));

1021         return (opendir(buf));
1022 }

1024 static int
1025 g_lstat(Char *fn, struct stat *sb, glob_t *pglob)
1026 {
1027         char buf[MAXPATHLEN];

1029         if (g_Ctoc(fn, buf, sizeof (buf)))
1030                 return (-1);
1031         if (pglob->gl_flags & GLOB_ALTDIRFUNC)
1032                 return ((*pglob->gl_lstat)(buf, sb));
1033         return (lstat(buf, sb));
1034 }

1036 static int
1037 g_stat(Char *fn, struct stat *sb, glob_t *pglob)
1038 {
1039         char buf[MAXPATHLEN];

1041         if (g_Ctoc(fn, buf, sizeof (buf)))
1042                 return (-1);
1043         if (pglob->gl_flags & GLOB_ALTDIRFUNC)
1044                 return ((*pglob->gl_stat)(buf, sb));
1045         return (stat(buf, sb));
1046 }

1048 static Char *
1049 g_strchr(const Char *str, int ch)
1050 {
1051         do {
1052                 if (*str == ch)
1053                         return ((Char *)str);
1054         } while (*str++);
1055         return (NULL);
1056 }

1058 static int
1059 g_Ctoc(const Char *str, char *buf, uint_t len)
1060 {

1062         while (len--) {
1063                 if ((*buf++ = *str++) == EOS)
1064                         return (0);
1065         }
1066         return (1);
```



```
1067 }

1069 #ifdef DEBUG
1070 static void
1071 qprintf(const char *str, Char *s)
1072 {
1073         Char *p;

1075         (void) printf("%s:\n", str);
1076         for (p = s; *p; p++)
1077                 (void) printf("%c", CHAR(*p));
1078         (void) printf("\n");
1079         for (p = s; *p; p++)
1080                 (void) printf("%c", *p & M_PROTECT ? '"' : ' ');
1081         (void) printf("\n");
1082         for (p = s; *p; p++)
1083                 (void) printf("%c", ismeta(*p) ? '_' : ' ');
1084         (void) printf("\n");
1085 }
1086 #endif
```

```
**********************************************************
   17716 Thu Nov  1 09:19:13 2012
new/usr/src/man/man3c/glob.3c
1097 glob(3c) needs to support non-POSIX options
**********************************************************
```

```
'\" te
.\" Copyright (c) 1992, X/Open Company Limited. All Rights Reserved.
.\" Portions Copyright (c) 2003, Sun Microsystems, Inc.  All Rights Reserved.
.\" Portions Copyright (c) 2012, Gary Mills
.\" Copyright (c) 1992, X/Open Company Limited. All Rights Reserved.  Portions C
.\" 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
.\"
.\" $OpenBSD: glob.3,v 1.30 2012/01/20 07:09:42 tedu Exp $
.\"
.\" Copyright (c) 1989, 1991, 1993, 1994
.\" The Regents of the University of California.  All rights reserved.
.\"
.\" This code is derived from software contributed to Berkeley by
.\" Guido van Rossum.
.\" Redistribution and use in source and binary forms, with or without
.\" modification, are permitted provided that the following conditions
.\" are met:
.\" 1. Redistributions of source code must retain the above copyright
.\"    notice, this list of conditions and the following disclaimer.
.\" 2. Redistributions in binary form must reproduce the above copyright
.\"    notice, this list of conditions and the following disclaimer in the
.\"    documentation and/or other materials provided with the distribution.
.\" 3. Neither the name of the University nor the names of its contributors
.\"    may be used to endorse or promote products derived from this software
.\"    without specific prior written permission.
.\"
.\" THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ''AS IS'' AND
.\" ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
.\" IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
.\" ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
.\" FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
.\" DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
.\" OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
.\" HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
.\" LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
.\" OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
.\" SUCH DAMAGE.
.\"
.\"  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 GLOB 3C "Nov 1, 2003"
.SH NAME
glob, globfree \- generate path names matching a pattern
.SH SYNOPSIS
.LP
.nf
#include <glob.h>

\fBint\fR \fBglob\fR(\fBconst char *restrict\fR \fIpattern\fR, \fBint\fR \fIflag
     \fBint(*\fR\fIerrfunc\fR)(const char *\fIepath\fR, int \fIeerrno)\fR,
     \fBglob_t *restrict\fR \fIpglob\fR);
.fi

.LP
.nf
\fBvoid\fR \fBglobfree\fR(\fBglob_t *\fR\fIpglob\fR);
.fi
```



```
.SH DESCRIPTION
.sp
.LP
The \fBglob()\fR function is a path name generator.
.sp
.LP
The \fBglobfree()\fR function frees any memory allocated by \fBglob()\fR
associated with \fIpglob\fR.
.SS "\fIpattern\fR Argument"
.sp
.LP
The argument \fIpattern\fR is a pointer to a path name pattern to be expanded.
The \fBglob()\fR function matches all accessible path names against this
pattern and develops a list of all path names that match. In order to have
access to a path name, \fBglob()\fR requires search permission on every
component of a path except the last, and read permission on each directory of
any filename component of \fIpattern\fR that contains any of the following
special characters:
.sp
.in +2
.nf
*        ?        [
.fi
.in -2

.SS "\fIpglob\fR Argument"
.sp
.LP
The structure type \fBglob_t\fR is defined in the header \fB<glob.h>\fR and
includes at least the following members:
.sp
.in +2
.nf
size_t   gl_pathc;     /* Total count of paths matched by */
size_t   gl_pathc;     /* count of paths matched by */
                       /* pattern */
char     **gl_pathv;   /* List of matched path names */
size_t   gl_offs;      /* # of slots reserved in gl_pathv */
int      gl_matchc;    /* Count of paths matching pattern. */
int      gl_flags;     /* Copy of flags parameter to glob. */
char     **gl_pathv;   /* pointer to list of matched */
                       /* path names */
size_t   gl_offs;      /* slots to reserve at beginning */
                       /* of gl_pathv */
.fi
.in -2

.sp
.LP
The \fBglob()\fR function stores the number of matched path names into
\fIpglob\(mi>\fR\fBgl_pathc\fR and a pointer to a list of pointers to path
names into \fIpglob\(mi>\fR\fBgl_pathv.\fR The path names are in sort order as
defined by the current setting of the  \fBLC_COLLATE\fR category. The first
pointer after the last path name is a \fINULL\fR pointer. If the pattern does
not match any path names, the returned number of matched paths is set to 0, and
the contents of \fIpglob\(mi>\fR\fBgl_pathv\fR are implementation-dependent.
.sp
.LP
It is the caller's responsibility to create the structure pointed to by
\fIpglob\fR. The \fBglob()\fR function allocates other space as needed,
including the memory pointed to by \fBgl_pathv\fR. The \fBglobfree()\fR
function frees any space associated with \fIpglob\fR from a previous call to
\fBglob()\fR.
.SS "\fIflags\fR Argument"
.sp
```

```
.LP
The \fIflags\fR argument is used to control the behavior of \fBglob()\fR. The
value of \fIflags\fR is a bitwise inclusive \fBOR\fR of zero or more of the
following constants, which are defined in the header <\fBglob.h\fR>:
.sp
.ne 2
.na
\fB\fBGLOB_APPEND\fR\fR
.ad
.RS 17n
Append path names generated to the ones from a previous call to \fBglob()\fR.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_DOOFFS\fR\fR
.ad
.RS 17n
Make use of \fIpglob\(mi>\fR\fBgl_offs\fR\fI\&.\fR If this flag is set,
\fIpglob\(mi>\fR\fBgl_offs\fR is used to specify how many \fINULL\fR pointers
to add to the beginning of \fIpglob\(mi>\fR\fBgl_pathv\fR\fI\&.\fR In other
words, \fIpglob\(mi>\fR\fBgl_pathv\fR will point to
\fIpglob\(mi>\fR\fBgl_offs\fR \fINULL\fR pointers, followed by
\fIpglob\(mi>\fR\fBgl_pathc\fR path name pointers, followed by a \fINULL\fR
pointer.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_ERR\fR\fR
.ad
.RS 17n
Causes \fBglob()\fR to return when it encounters a directory that it cannot
open or read. Ordinarily, \fBglob()\fR continues to find matches.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_MARK\fR\fR
.ad
.RS 17n
Each path name that is a directory that matches \fIpattern\fR has a slash
appended.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_NOCHECK\fR\fR
.ad
.RS 17n
If \fIpattern\fR does not match any path name, then \fBglob()\fR returns a list
consisting of only \fIpattern\fR, and the number of matched path names is 1.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_NOESCAPE\fR\fR
.ad
.RS 17n
Disable backslash escaping.
.RE
```



```
.sp
.ne 2
.na
\fB\fBGLOB_NOSORT\fR\fR
.ad
.RS 17n
Ordinarily, \fBglob()\fR sorts the matching path names according to the current
setting of the \fBLC_COLLATE\fR category.  When this flag is used the order of
path names returned is unspecified.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_ALTDIRFUNC\fR\fR
.ad
.RS 17n
The following additional fields in the \fIpglob\fR structure
have been initialized with alternate functions for
\fBglob()\fR to use to open, read, and close directories and
to get stat information on names found in those directories:
.sp
.nf
DIR *(*gl_opendir)(const char *);
struct dirent *(*gl_readdir)(DIR *);
void (*gl_closedir)(DIR *);
int (*gl_lstat)(const char *, struct stat *);
int (*gl_stat)(const char *, struct stat *);
.fi
.sp
This extension is provided to allow programs such as
\fBufsrestore\fR(1M) to provide globbing from directories stored
on tape.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_BRACE\fR\fR
.ad
.RS 17n
Pre-process the pattern string to expand `{pat,pat,...}'
strings like \fBcsh\fR(1).  The pattern `{}' is left unexpanded
for historical reasons.  (\fBcsh\fR(1) does the same thing
to ease typing of \fBfind\fR(1) patterns.)
.RE

.sp
.ne 2
.na
\fB\fBGLOB_MAGCHAR\fR\fR
.ad
.RS 17n
Set by the \fBglob()\fR function if the pattern included globbing
characters.  See the description of the usage of
the \fBgl_matchc\fR structure member for more details.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_NOMAGIC\fR\fR
.ad
.RS 17n
Is the same as \fBGLOB_NOCHECK\fR but it only appends the
```

```
pattern if it does not contain any of the special characters
‘*’, ‘?’, or ‘[’.  \fBGLOB_NOMAGIC\fR is provided to
simplify implementing the historic \fBcsh\fR(1) globbing behavior
and should probably not be used anywhere else.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_QUOTE\fR\fR
.ad
.RS 17n
This option has no effect and is included for backwards
compatibility with older sources.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_TILDE\fR\fR
.ad
.RS 17n
Expand patterns that start with ‘~’ to user name home
directories.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_LIMIT\fR\fR
.ad
.RS 17n
Limit the amount of memory used by matches to \fIARG_MAX\fR.
This option should be set for programs that can be coerced
to a denial of service attack via patterns that
expand to a very large number of matches, such as a long
string of ‘*/../*/..’.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_KEEPSTAT\fR\fR
.ad
.RS 17n
Retain a copy of the \fBstat\fR(2) information retrieved for
matching paths in the gl_statv array:
.sp
.nf
struct stat **gl_statv;
.fi
.sp
This option may be used to avoid \fBlstat\fR(2) lookups in
cases where they are expensive.
.RE

.sp
.LP
The \fBGLOB_APPEND\fR flag can be used to append a new set of path names to
those found in a previous call to \fBglob()\fR. The following rules apply when
two or more calls to \fBglob()\fR are made with the same value of \fIpglob\fR
and without intervening calls to \fBglobfree()\fR:
.RS +4
.TP
1.
The first such call must not set \fBGLOB_APPEND.\fR All subsequent calls
```



```
must set it.
.RE
.RS +4
.TP
2.
All the calls must set \fBGLOB_DOOFFS,\fR or all must not set it.
.RE
.RS +4
.TP
3.
After the second call, \fIpglob\(mi>\fR\fBgl_pathv\fR points to a list
containing the following:
.RS +4
.TP
a.
Zero or more \fINULL\fR pointers, as specified by \fBGLOB_DOOFFS\fR and
\fIpglob\(mi>\fR\fBgl_offs\fR.
.RE
.RS +4
.TP
b.
Pointers to the path names that were in the \fIpglob\(mi>\fR\fBgl_pathv\fR
list before the call, in the same order as before.
.RE
.RS +4
.TP
c.
Pointers to the new path names generated by the second call, in the
specified order.
.RE
.RE
.RS +4
.TP
4.
The count returned in \fIpglob\(mi>\fR\fBgl_pathc\fR will be the total
number of path names from the two calls.
.RE
.RS +4
.TP
5.
The application can change any of the fields after a call to \fBglob()\fR.
If it does, it must reset them to the original value before a subsequent call,
using the same \fIpglob\fR value, to \fBglobfree()\fR or \fBglob()\fR with the
\fBGLOB_APPEND\fR flag.
.RE
.SS "\fIerrfunc\fR and \fIepath\fR Arguments"
.sp
.LP
If, during the search, a directory is encountered that cannot be opened or read
and \fIerrfunc\fR is not a \fINULL\fR pointer, \fBglob()\fR calls
\fB(\fR\fI*errfunc\fR\fB)\fR with two arguments:
.RS +4
.TP
1.
The \fIepath\fR argument is a pointer to the path that failed.
.RE
.RS +4
.TP
2.
The \fIeerrno\fR argument is the value of \fIerrno\fR from the failure, as
set by the \fBopendir\fR(3C), \fBreaddir\fR(3C) or \fBstat\fR(2) functions.
(Other values may be used to report other errors not explicitly documented for
those functions.)
.RE

.sp
```

```
.LP
If \fB(\fR\fI*errfunc\fR\fB)\fR is called and returns non-zero, or if the
\fBGLOB_ERR\fR flag is set in \fIflags\fR, \fBglob()\fR stops the scan and
returns \fBGLOB_ABORTED\fR after setting \fIgl_pathc\fR and \fIgl_pathv\fR in
\fIpglob\fR to reflect the paths already scanned. If \fBGLOB_ERR\fR is not set
and either \fIerrfunc\fR is a \fINULL\fR pointer or
\fB(\fR\fI*errfunc\fR\fB)\fR returns 0, the error is ignored.
.SH RETURN VALUES
The following constants are defined as error return values for \fBglob()\fR:
.sp
.LP
On successful completion, \fBglob()\fR returns zero.
In addition the fields of pglob contain the values described below:

.sp
.ne 2
.na
\fB\fBgl_pathc\fR\fR
\fB\fBGLOB_ABORTED\fR\fR
.ad
.RS 16n
Contains the total number of matched pathnames so far.
This includes other matches from previous invocations of
\fBglob()\fR if \fBGLOB_APPEND\fR was specified.
The scan was stopped because \fBGLOB_ERR\fR was set or
\fB(\fR\fI*errfunc\fR\fB)\fR returned non-zero.
.RE

.sp
.ne 2
.na
\fB\fBgl_matchc\fR\fR
\fB\fBGLOB_NOMATCH\fR\fR
.ad
.RS 16n
Contains the number of matched pathnames in the current
invocation of \fBglob()\fR.
The pattern does not match any existing path name, and \fBGLOB_NOCHECK\fR was
not set in flags.
.RE

.sp
.ne 2
.na
\fB\fBgl_flags\fR\fR
\fB\fBGLOG_NOSPACE\fR\fR
.ad
.RS 16n
Contains a copy of the flags parameter with the bit
\fBGLOB_MAGCHAR\fR set if pattern contained any of the special
characters ‘*’, ‘?’, or ‘[’, cleared if not.
An attempt to allocate memory failed.
.RE

.sp
.ne 2
.na
\fB\fBgl_pathv\fR\fR
.ad
.RS 16n
Contains a pointer to a null-terminated list of matched
pathnames.  However, if \fBgl_pathc\fR is zero, the contents of
\fBgl_pathv\fR are undefined.
.RE

.LP
```



```
If \fB(\fR\fI*errfunc\fR\fB)\fR is called and returns non-zero, or if the
\fBGLOB_ERR\fR flag is set in \fIflags\fR, \fBglob()\fR stops the scan and
returns \fBGLOB_ABORTED\fR after setting \fIgl_pathc\fR and \fIgl_pathv\fR in
\fIpglob\fR to reflect the paths already scanned. If \fBGLOB_ERR\fR is not set
and either \fIerrfunc\fR is a \fINULL\fR pointer or
\fB(\fR\fI*errfunc\fR\fB)\fR returns 0, the error is ignored.
.SH RETURN VALUES
.sp
.ne 2
.na
\fB\fBgl_statv\fR\fR
.ad
.RS 16n
If the \fBGLOB_KEEPSTAT\fR flag was set, \fBgl_statv\fR contains a
pointer to a null-terminated list of matched \fBstat\fR(2)
objects corresponding to the paths in \fBgl_pathc\fR.
.RE

.sp
.LP
If \fBglob()\fR terminates due to an error, it sets \fBerrno\fR and
returns one of the following non-zero constants. defined in <\fBglob.h\fR>:

The following values are returned by \fBglob()\fR:
.sp
.ne 2
.na
\fB\fBGLOB_ABORTED\fR\fR
\fB\fB0\fR\fR
.ad
.RS 16n
The scan was stopped because \fBGLOB_ERR\fR was set or
\fB(\fR\fI*errfunc\fR\fB)\fR returned non-zero.
.RS 12n
Successful completion. The argument \fIpglob\(mi>\fR\fBgl_pathc\fR returns the
number of matched path names and the argument \fIpglob\(mi>\fR\fBgl_pathv\fR
contains a pointer to a null-terminated list of matched and sorted path names.
However, if \fIpglob\(mi>\fR\fBgl_pathc\fR is 0, the content of
\fIpglob\(mi>\fR\fBgl_pathv\fR is undefined.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_NOMATCH\fR\fR
\fB\fBnon-zero\fR\fR
.ad
.RS 16n
The pattern does not match any existing path name, and \fBGLOB_NOCHECK\fR was
not set in flags.
.RS 12n
An error has occurred. Non-zero constants are defined in <\fBglob.h\fR>. The
arguments \fIpglob\(mi>\fR\fBgl_pathc\fR and \fIpglob\(mi>\fR\fBgl_pathv\fR are
still set as defined above.
.RE

.sp
.ne 2
.na
\fB\fBGLOB_NOSPACE\fR\fR
.ad
.RS 16n
An attempt to allocate memory failed.
.RE

.sp
```

```
.ne 2
.na
\fB\fBGLOB_NOSYS\fR\fR
.ad
.RS 16n
The requested function is not supported by this version of
\fBglob()\fR.
.RE

.LP
The arguments \fIpglob\(mi>\fR\fBgl_pathc\fR and \fIpglob\(mi>\fR\fBgl_pathv\fR
specified above.
.sp
.LP
The \fBglobfree()\fR function returns no value.
.SH USAGE
.sp
.LP
This function is not provided for the purpose of enabling utilities to perform
path name expansion on their arguments, as this operation is performed by the
shell, and utilities are explicitly not expected to redo this. Instead, it is
provided for applications that need to do path name expansion on strings
obtained from other sources, such as a pattern typed by a user or read from a
file.
.sp
.LP
If a utility needs to see if a path name matches a given pattern, it can use
\fBfnmatch\fR(3C).
.sp
.LP
Note that \fBgl_pathc\fR and \fBgl_pathv\fR have meaning even if \fBglob()\fR
fails. This allows \fBglob()\fR to report partial results in the event of an
error. However, if \fBgl_pathc\fR is 0, \fBgl_pathv\fR is unspecified even if
\fBglob()\fR did not return an error.
.sp
.LP
The \fBGLOB_NOCHECK\fR option could be used when an application wants to expand
a path name if wildcards are specified, but wants to treat the pattern as just
a string otherwise.
.sp
.LP
The new path names generated by a subsequent call with \fBGLOB_APPEND\fR are
not sorted together with the previous path names. This mirrors the way that the
shell handles path name expansion when multiple expansions are done on a
command line.
.sp
.LP
Applications that need tilde and parameter expansion should use the
\fBwordexp\fR(3C) function.
.SH EXAMPLES
.LP
\fBExample 1 \fRExample of \fBglob_doofs\fR function.
.sp
.LP
One use of the \fBGLOB_DOOFFS\fR flag is by applications that build an argument
list for use with the \fBexecv()\fR, \fBexecve()\fR, or \fBexecvp()\fR
functions (see \fBexec\fR(2)). Suppose, for example, that an application wants
to do the equivalent of:

.sp
.in +2
.nf
\fBls\fR \fB-l\fR *.c
.fi
.in -2
```



```
.sp
.LP
but for some reason:

.sp
.in +2
.nf
system("ls -l *.c")
.fi
.in -2

.sp
.LP
is not acceptable. The application could obtain approximately the same result
using the sequence:

.sp
.in +2
.nf
globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &globbuf);
globbuf.gl_pathv[0] = "ls";
globbuf.gl_pathv[1] = "-l";
execvp ("ls", &globbuf.gl_pathv[0]);
.fi
.in -2

.sp
.LP
Using the same example:

.sp
.in +2
.nf
\fBls\fR \fB-l\fR *.c *.h
.fi
.in -2

.sp
.LP
could be approximately simulated using \fBGLOB_APPEND\fR as follows:

.sp
.in +2
.nf
\fBglobbuf.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &globbuf);
glob ("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
\&.\|.\|.\fR
.fi
.in -2

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
Interface Stability     Standard
```

```
620 _
621 MT-Level        MT-Safe
622 .TE

624 .SH SEE ALSO
625 .sp
626 .LP
627 \fBexecv\fR(2), \fBstat\fR(2), \fBfnmatch\fR(3C), \fBopendir\fR(3C),
628 \fBreaddir\fR(3C), \fBwordexp\fR(3C), \fBattributes\fR(5), \fBstandards\fR(5)
```