Walk a directory/Recursively

From Rosetta Code

Walk a given directory *tree* and print files matching a given pattern.

Note: This task is for recursive methods. These tasks should read an entire directory tree, not a *single directory*. For code examples that read a *single directory*, see Walk a directory/Non-recursively.

Note: Please be careful when running any code examples found here.

Walk a directory/Recursively
You are encouraged to solve this task according to the task description, using any language you may know.

Contents

Ada

```
with Ada.Directories;  use Ada.Directories;
with Ada.Text_IO;

procedure Test_Directory_Walk is
   procedure Walk (Name : String; Pattern : String) is
      procedure Print (Item : Directory_Entry_Type) is
      begin
         Ada.Text_IO.Put_Line (Full_Name (Item));
      end Print;
      procedure Walk (Item : Directory_Entry_Type) is
      begin
         if Simple_Name (Item) /= "." and then Simple_Name (Item) /= ".." then
            Walk (Full_Name (Item), Pattern);
         end if;
      exception
         when Name_Error => null;
      end Walk;
   begin
      Search (Name, Pattern, (others => True), Print'Access);
      Search (Name, "", (Directory => True, others => False), Walk'Access);
   end Walk;
begin
   Walk (".", "*.adb");
end Test_Directory_Walk;
```

The solution first enumerates files in a directory, that includes the subdirectories, if their names match the pattern. Then it steps down into each of the subdirectories. The pseudo directories . and .. are excluded. The behavior upon symbolic links depends on the OS and the implementation of the Ada.Directories package.

ALGOL 68

Works with: ALGOL 68G version Any - tested with release mk15-0.8b.fc9.i386 - uses non-standard library routines *get directory* and *grep in string*.

```
INT match=0, no match=1, out of memory error=2, other error=3;

STRING slash = "/", pwd=".", parent="..";

PROC walk tree = (STRING path, PROC (STRING)VOID call back)VOID: (
  []STRING files = get directory(path);
  FOR file index TO UPB files DO
    STRING file = files[file index];
    STRING path file = path+slash+file;
    IF file is directory(path file) THEN
      IF file NE pwd AND file NE parent THEN
        walk tree(path file, call back)
      FI
    ELSE
      call back(path file)
    FI
  OD
);

STRING re sort a68 = "[Ss]ort[^/]*[.]a68$";

PROC match sort a68 and print = (STRING path file)VOID:
  IF grep in string(re sort a68, path file, NIL, NIL) = match THEN
    print((path file, new line))
  FI;

walk tree(".", match sort a68 and print)
```

Sample output:

```
./Shell_sort_c.a68
./Quick_sort.a68
./Shell_sort.a68
./Cocktail_Sort.a68
./Selection_Sort.a68
./Merge_sort.a68
./tmp/test_sort.a68
./Bobosort.a68
./Sorting_an_Array_of_Integers.a68
./Insertion_Sort.a68
./Permutation_Sort.a68
```

AutoHotkey

Display all TMP files in Temp directory and its subdirectories.

```
Loop, %A_Temp%\*.tmp,,1
 out .= A_LoopFileName "`n"
MsgBox,% out
```

Batch File

```
dir /a-d %1
```

If you wanted to apply some command to each item in a directory tree, then use `FOR` with the switch `/R`. For example, to apply the ECHO command to every DLL file in C:\Windows\System32:

Works with: Windows NT version 4 or later (includes Windows XP and onward)

```
FOR /R C:\Windows\System32 %%F IN (*.DLL) DO ECHO "%%F"
```

This can be done from outside a batch file (entered directly at the command prompt) by changing the double percent signs (%%) to single percents (%):

```
FOR /R C:\Windows\System32 %F IN (*.DLL) DO ECHO "%F"
```

BBC BASIC

Works with: BBC BASIC for Windows

```
      directory$ = "C:\Windows\"
      pattern$ = "*.chm"
      PROClisttree(directory$, pattern$)
      END
      
      DEF PROClisttree(dir$, filter$)
      LOCAL dir%, sh%, res%
      DIM dir% LOCAL 317
      IF RIGHT$(dir$) <> "\" IF RIGHT$(dir$) <> "/" dir$ += "\"
      SYS "FindFirstFile", dir$ + filter$, dir% TO sh%
      IF sh% <> -1 THEN
        REPEAT
          IF (!dir% AND 16) = 0 PRINT dir$ + $$(dir%+44)
          SYS "FindNextFile", sh%, dir% TO res%
        UNTIL res% = 0
        SYS "FindClose", sh%
      ENDIF
      SYS "FindFirstFile", dir$ + "*", dir% TO sh%
      IF sh% <> -1 THEN
        REPEAT
          IF (!dir% AND 16) IF dir%?44 <> &2E THEN
            PROClisttree(dir$ + $$(dir%+44) + "\", filter$)
          ENDIF
          SYS "FindNextFile", sh%, dir% TO res%
        UNTIL res% = 0
        SYS "FindClose", sh%
      ENDIF
      ENDPROC
```

C

Library: POSIX

Works with: POSIX version .1-2001

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>
#include <regex.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <err.h>

enum {
        WALK_OK = 0,
        WALK_BADPATTERN,
        WALK_NAMETOOLONG,
        WALK_BADIO,
};

#define WS_NONE         0
#define WS_RECURSIVE    (1 << 0)
#define WS_DEFAULT      WS_RECURSIVE
#define WS_FOLLOWLINK   (1 << 1)        /* follow symlinks */
#define WS_DOTFILES     (1 << 2)        /* per unix convention, .file is hidden */
#define WS_MATCHDIRS    (1 << 3)        /* if pattern is used on dir names too */

int walk_recur(char *dname, regex_t *reg, int spec)
{
        struct dirent *dent;
        DIR *dir;
        struct stat st;
        char fn[FILENAME_MAX];
        int res = WALK_OK;
        int len = strlen(dname);
        if (len >= FILENAME_MAX - 1)
                return WALK_NAMETOOLONG;

        strcpy(fn, dname);
        fn[len++] = '/';

        if (!(dir = opendir(dname))) {
                warn("can't open %s", dname);
                return WALK_BADIO;
        }

        errno = 0;
        while ((dent = readdir(dir))) {
                if (!(spec & WS_DOTFILES) && dent->d_name[0] == '.')
                        continue;
                if (!strcmp(dent->d_name, ".") || !strcmp(dent->d_name, ".."))
                        continue;

                strncpy(fn + len, dent->d_name, FILENAME_MAX - len);
                if (lstat(fn, &st) == -1) {
                        warn("Can't stat %s", fn);
                        res = WALK_BADIO;
                        continue;
                }

                /* don't follow symlink unless told so */
                if (S_ISLNK(st.st_mode) && !(spec & WS_FOLLOWLINK))
                        continue;

                /* will be false for symlinked dirs */
                if (S_ISDIR(st.st_mode)) {
                        /* recursively follow dirs */
                        if ((spec & WS_RECURSIVE))
                                walk_recur(fn, reg, spec);

                        if (!(spec & WS_MATCHDIRS)) continue;
                }

                /* pattern match */
                if (!regexec(reg, fn, 0, 0, 0)) puts(fn);
        }

        if (dir) closedir(dir);
        return res ? res : errno ? WALK_BADIO : WALK_OK;
}

int walk_dir(char *dname, char *pattern, int spec)
{
        regex_t r;
        int res;
        if (regcomp(&r, pattern, REG_EXTENDED | REG_NOSUB))
                return WALK_BADPATTERN;
        res = walk_recur(dname, &r, spec);
        regfree(&r);

        return res;
}

int main()
{
        int r = walk_dir(".", ".\\.c$", WS_DEFAULT|WS_MATCHDIRS);
        switch(r) {
        case WALK_OK:           break;
        case WALK_BADIO:        err(1, "IO error");
        case WALK_BADPATTERN:   err(1, "Bad pattern");
        case WALK_NAMETOOLONG:  err(1, "Filename too long");
        default:
                err(1, "Unknown error?");
        }
        return 0;
}
```

Library: BSD libc

With the fts(3) (http://www.openbsd.org/cgi-bin/man.cgi?query=fts&apropos=0&sektion=3&manpath=OpenBSD+Current&arch=i386&format=html) functions from 4.4BSD, this program can sort the files, and can also detect cycles (when a link puts a directory inside itself). This program makes a *logical traversal* that follows symbolic links to directories.

Works with: OpenBSD version 4.9

```
#include <sys/types.h>
#include <err.h>
#include <errno.h>
#include <fnmatch.h>
#include <fts.h>
#include <string.h>
#include <stdio.h>

/* Compare files by name. */
int
entcmp(const FTSENT **a, const FTSENT **b)
{
        return strcmp((*a)->fts_name, (*b)->fts_name);
}

/*
 * Print all files in the directory tree that match the glob pattern.
 * Example: pmatch("/usr/src", "*.c");
 */
void
pmatch(char *dir, const char *pattern)
{
        FTS *tree;
        FTSENT *f;
        char *argv[] = { dir, NULL };

        /*
         * FTS_LOGICAL follows symbolic links, including links to other
         * directories. It detects cycles, so we never have an infinite
         * loop. FTS_NOSTAT is because we never use f->statp. It uses
         * our entcmp() to sort files by name.
         */
        tree = fts_open(argv, FTS_LOGICAL | FTS_NOSTAT, entcmp);
        if (tree == NULL)
                err(1, "fts_open");

        /*
         * Iterate files in tree. This iteration always skips
         * "." and ".." because we never use FTS_SEEDOT.
         */
        while ((f = fts_read(tree))) {
                switch (f->fts_info) {
                case FTS_DNR:   /* Cannot read directory */
                case FTS_ERR:   /* Miscellaneous error */
                case FTS_NS:    /* stat() error */
                        /* Show error, then continue to next files. */
                        warn("%s", f->fts_path);
                        continue;
                case FTS_DP:
                        /* Ignore post-order visit to directory. */
                        continue;
                }

                /*
                 * Check if name matches pattern. If so, then print
                 * path. This check uses FNM_PERIOD, so "*.c" will not
                 * match ".invisible.c".
                 */
                if (fnmatch(pattern, f->fts_name, FNM_PERIOD) == 0)
                        puts(f->fts_path);

                /*
                 * A cycle happens when a symbolic link (or perhaps a
                 * hard link) puts a directory inside itself. Tell user
                 * when this happens.
                 */
                if (f->fts_info == FTS_DC)
                        warnx("%s: cycle in directory tree", f->fts_path);
        }

        /* fts_read() sets errno = 0 unless it has error. */
        if (errno != 0)
                err(1, "fts_read");

        if (fts_close(tree) < 0)
                err(1, "fts_close");
}

int
main()
{
        pmatch(".", "*.c");
        return 0;
}
```

Windows

Library: Win32
Works with: MinGW

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>

/* Print "message: last Win32 error" to stderr. */
void
oops(const wchar_t *message)
{
        wchar_t *buf;
        DWORD error;

        buf = NULL;
        error = GetLastError();
        FormatMessageW(FORMAT_MESSAGE_ALLOCATE_BUFFER |
            FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
            NULL, error, 0, (wchar_t *)&buf, 0, NULL);

        if (buf) {
                fwprintf(stderr, L"%ls: %ls", message, buf);
                LocalFree(buf);
        } else {
                /* FormatMessageW failed. */
                fwprintf(stderr, L"%ls: unknown error 0x%x\n",
                    message, error);
        }
}

/*
 * Print all files in a given directory tree that match a given wildcard
 * pattern.
 */
int
main()
{
        struct stack {
                wchar_t                 *path;
                size_t                   pathlen;
                size_t                   slashlen;
                HANDLE                   ffh;
                WIN32_FIND_DATAW         ffd;
                struct stack            *next;
        } *dir, dir0, *ndir;
        size_t patternlen;
        int argc;
        wchar_t **argv, *buf, c, *pattern;

        /* MinGW never provides wmain(argc, argv). */
        argv = CommandLineToArgvW(GetCommandLineW(), &argc);
        if (argv == NULL) {
                oops(L"CommandLineToArgvW");
                exit(1);
        }

        if (argc != 3) {
                fwprintf(stderr, L"usage: %ls dir pattern\n", argv[0]);
                exit(1);
        }

        dir0.path = argv[1];
        dir0.pathlen = wcslen(dir0.path);
        pattern = argv[2];
        patternlen = wcslen(pattern);

        if (patternlen == 0 ||
            wcscmp(pattern, L".") == 0 ||
            wcscmp(pattern, L"..") == 0 ||
            wcschr(pattern, L'/') ||
            wcschr(pattern, L'\\')) {
                fwprintf(stderr, L"%ls: invalid pattern\n", pattern);
                exit(1);
        }

        /*
         * Must put backslash between path and pattern, unless
         * last character of path is slash or colon.
         *
         *   'dir' => 'dir\*'
         *   'dir\' => 'dir\*'
         *   'dir/' => 'dir/*'
         *   'c:' => 'c:*'
         *
         * 'c:*' and 'c:\*' are different files!
         */
        c = dir0.path[dir0.pathlen - 1];
        if (c == ':' || c == '/' || c == '\\')
                dir0.slashlen = dir0.pathlen;
        else
```

```
                dir0.slashlen = dir0.pathlen + 1;

        /* Allocate space for path + backslash + pattern + \0. */
        buf = calloc(dir0.slashlen + patternlen + 1, sizeof buf[0]);
        if (buf == NULL) {
                perror("calloc");
                exit(1);
        }
        dir0.path = wmemcpy(buf, dir0.path, dir0.pathlen + 1);

        dir0.ffh = INVALID_HANDLE_VALUE;
        dir0.next = NULL;
        dir = &dir0;

        /* Loop for each directory in linked list. */
loop:
        while (dir) {
                /*
                 * At first visit to directory:
                 *   Print the matching files. Then, begin to find
                 *   subdirectories.
                 *
                 * At later visit:
                 *   dir->ffh is the handle to find subdirectories.
                 *   Continue to find them.
                 */
                if (dir->ffh == INVALID_HANDLE_VALUE) {
                        /* Append backslash + pattern + \0 to path. */
                        dir->path[dir->pathlen] = '\\';
                        wmemcpy(dir->path + dir->slashlen,
                            pattern, patternlen + 1);

                        /* Find all files to match pattern. */
                        dir->ffh = FindFirstFileW(dir->path, &dir->ffd);
                        if (dir->ffh == INVALID_HANDLE_VALUE) {
                                /* Check if no files match pattern. */
                                if (GetLastError() == ERROR_FILE_NOT_FOUND)
                                        goto subdirs;

                                /* Bail out from other errors. */
                                dir->path[dir->pathlen] = '\0';
                                oops(dir->path);
                                goto popdir;
                        }

                        /* Remove pattern from path; keep backslash. */
                        dir->path[dir->slashlen] = '\0';

                        /* Print all files to match pattern. */
                        do {
                                wprintf(L"%ls%ls\n",
                                    dir->path, dir->ffd.cFileName);
                        } while (FindNextFileW(dir->ffh, &dir->ffd) != 0);
                        if (GetLastError() != ERROR_NO_MORE_FILES) {
                                dir->path[dir->pathlen] = '\0';
                                oops(dir->path);
                        }
                        FindClose(dir->ffh);

subdirs:
                        /* Append * + \0 to path. */
                        dir->path[dir->slashlen] = '*';
                        dir->path[dir->slashlen + 1] = '\0';

                        /* Find first possible subdirectory. */
                        dir->ffh = FindFirstFileExW(dir->path,
                            FindExInfoStandard, &dir->ffd,
                            FindExSearchLimitToDirectories, NULL, 0);
                        if (dir->ffh == INVALID_HANDLE_VALUE) {
                                dir->path[dir->pathlen] = '\0';
                                oops(dir->path);
                                goto popdir;
                        }
                } else {
                        /* Find next possible subdirectory. */
```

```
				if (FindNextFileW(dir->ffh, &dir->ffd) == 0)
					goto closeffh;
			}

			/* Enter subdirectories. */
			do {
				const wchar_t *fn = dir->ffd.cFileName;
				const DWORD attr = dir->ffd.dwFileAttributes;
				size_t buflen, fnlen;

				/*
				 * Skip '.' and '..', because they are links to
				 * the current and parent directories, so they
				 * are not subdirectories.
				 *
				 * Skip any file that is not a directory.
				 *
				 * Skip all reparse points, because they might
				 * be symbolic links. They might form a cycle,
				 * with a directory inside itself.
				 */
				if (wcscmp(fn, L".") == 0 ||
				    wcscmp(fn, L"..") == 0 ||
				    (attr & FILE_ATTRIBUTE_DIRECTORY) == 0 ||
				    (attr & FILE_ATTRIBUTE_REPARSE_POINT))
					continue;

				ndir = malloc(sizeof *ndir);
				if (ndir == NULL) {
					perror("malloc");
					exit(1);
				}

				/*
				 * Allocate space for path + backslash +
				 *     fn + backslash + pattern + \0.
				 */
				fnlen = wcslen(fn);
				buflen = dir->slashlen + fnlen + patternlen + 2;
				buf = calloc(buflen, sizeof buf[0]);
				if (buf == NULL) {
					perror("malloc");
					exit(1);
				}

				/* Copy path + backslash + fn + \0. */
				wmemcpy(buf, dir->path, dir->slashlen);
				wmemcpy(buf + dir->slashlen, fn, fnlen + 1);

				/* Push dir to list. Enter dir. */
				ndir->path = buf;
				ndir->pathlen = dir->slashlen + fnlen;
				ndir->slashlen = ndir->pathlen + 1;
				ndir->ffh = INVALID_HANDLE_VALUE;
				ndir->next = dir;
				dir = ndir;
				goto loop; /* Continue outer loop. */
			} while (FindNextFileW(dir->ffh, &dir->ffd) != 0);
closeffh:
			if (GetLastError() != ERROR_NO_MORE_FILES) {
				dir->path[dir->pathlen] = '\0';
				oops(dir->path);
			}
			FindClose(dir->ffh);

popdir:
			/* Pop dir from list, free dir, but never free dir0. */
			free(dir->path);
			if (ndir = dir->next)
				free(dir);
			dir = ndir;
		}

		return 0;
}
```

C#

```
using System.IO;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            DirectoryInfo tDir = new DirectoryInfo(@"C:\");
            string Pattern = "a";

            TraverseDirs(tDir, Pattern);
            Console.Read();
        }

        private static void TraverseDirs(DirectoryInfo dir, string Pattern)
        {
            // Subdirs
            try         // Avoid errors such as "Access Denied"
            {
                foreach (DirectoryInfo iInfo in dir.GetDirectories())
                {
                    if (iInfo.Name.StartsWith(Pattern))
                        Console.WriteLine("Found dir:  " + iInfo.FullName);

                    TraverseDirs(iInfo, Pattern);
                }
            }
            catch (Exception)
            {
            }

            // Subfiles
            try         // Avoid errors such as "Access Denied"
            {
                foreach (FileInfo iInfo in dir.GetFiles())
                {
                    if (iInfo.Name.StartsWith(Pattern))
                        Console.WriteLine("Found file: " + iInfo.FullName);
                }
            }
            catch (Exception)
            {
            }
        }
    }
}
```

C++

Library: boost

```
#include "boost/filesystem.hpp"
#include "boost/regex.hpp"
#include <iostream>

using namespace boost::filesystem;

int main()
{
  path current_dir("."); //
  boost::regex pattern("a.*"); // list all files starting with a
  for (recursive_directory_iterator iter(current_dir), end;
       iter != end;
```

```
      ++iter)
  {
    std::string name = iter->path().leaf();
    if (regex_match(name, pattern))
      std::cout << iter->path() << "\n";
  }
}
```

Caché ObjectScript

```

Class Utils.File [ Abstract ]
{

ClassMethod WalkTree(pDir As %String = "", pMask As %String = "*.*") As %Status
{
	// do some validation
	If pDir="" Quit $$$ERROR($$$GeneralError, "No directory specified.")

	// search input directory for files matching wildcard
	Set fs=##class(%ResultSet).%New("%File.FileSet")
	Set sc=fs.Execute(pDir, pMask)
	While (fs.Next()) {
		Write !, fs.Name
		// sub-directory
		If fs.Type="D" Set sc=..WalkTree(fs.Name, pMask)
	}

	// finished
	Quit $$$OK
}

}

```

Example:

```
USER>Do ##class(Utils.File).WalkTree("/Swsetup/")

C:\Swsetup\Monitors
C:\Swsetup\Monitors\HP_w2207_3.0
C:\Swsetup\Monitors\HP_w2207_3.0\Files
C:\Swsetup\Monitors\HP_w2207_3.0\Files\HP_w2207.cat
C:\Swsetup\Monitors\HP_w2207_3.0\Files\HP_w2207.icm
C:\Swsetup\Monitors\HP_w2207_3.0\Files\HP_w2207.inf
C:\Swsetup\Monitors\HP_w2207_3.0\HP Display Installer.exe
C:\Swsetup\Monitors\HP_w2207_3.0\HPx64DRV.exe
C:\Swsetup\Monitors\HP_w2207_3.0\Readme.txt
```

CoffeeScript

Works with: node.js

```
fs = require 'fs'

walk = (dir, f_match, f_visit) ->
  _walk = (dir) ->
    fns = fs.readdirSync dir
    for fn in fns
      fn = dir + '/' + fn
      if f_match fn
        f_visit fn
      if fs.statSync(fn).isDirectory()
        _walk fn
  _walk(dir)

dir = '..'
matcher = (fn) -> fn.match /\.coffee/
action = console.log
walk dir, matcher, action
```

Common Lisp

Library: CL-FAD

This example uses the `CL-FAD` library to achieve compatibility where the ANSI CL standard leaves ambiguities about pathnames.

```
(defun mapc-directory-tree (fn directory)
  (dolist (entry (cl-fad:list-directory directory))
    (when (cl-fad:directory-pathname-p entry)
      (mapc-directory-tree fn entry))
    (funcall fn entry)))
```

```
CL-USER> (mapc-directory-tree (lambda (x)
                                (when (equal (pathname-type x) "lisp")
                                  (write-line (namestring x))))
                              "lang/")
/home/sthalik/lang/lisp/.#bitmap.lisp
/home/sthalik/lang/lisp/avg.lisp
/home/sthalik/lang/lisp/bitmap.lisp
/home/sthalik/lang/lisp/box-muller.lisp
/home/sthalik/lang/lisp/displaced-subseq.lisp
[...]
```

Clojure

The standard function *file-seq* does a tree walk.

```
(use '[clojure.java.io])

(defn walk [dirpath pattern]
  (doall (filter #(re-matches pattern (.getName %))
                 (file-seq (file dirpath)))))

(map #(println (.getPath %)) (walk "src" #".*\.clj"))
```

D

```
void main() {
    import std.stdio, std.file;

    // Recursive breadth-first scan (use SpanMode.depth for
    // a depth-first scan):
    dirEntries("", "*.d", SpanMode.breadth).writeln;
}
```

E

```
def walkTree(directory, pattern) {
  for name => file in directory {
    if (name =~ rx`.*$pattern.*`) {
      println(file.getPath())
    }
    if (file.isDirectory()) {
      walkTree(file, pattern)
    }
  }
}
```

Example:

```
? walkTree(<file:/usr/share/man>, "rmdir")
/usr/share/man/man1/rmdir.1
/usr/share/man/man2/rmdir.2
```

Erlang

Use builtin function filelib:fold_files/5

Output:

```
10> filelib:fold_files( "/tmp", ".*", true, fun(File, Acc) -> [File|Acc] end, []).
["/tmp/clearcase_inst/local.conf", "/tmp/.X0-lock","/tmp/.cron-check-4000-was-here",
 "/tmp/kerneloops.XyN0SP","/tmp/npicagwD7tf"]
11> filelib:fold_files( "/tmp", ".*\.conf", true, fun(File, Acc) -> [File|Acc] end, []).
["/tmp/clearcase_inst/local.conf"]
```

F#

This code is tail-recursive and lazy.

```
open System.IO

let rec getAllFiles dir pattern =
    seq { yield! Directory.EnumerateFiles(dir, pattern)
          for d in Directory.EnumerateDirectories(dir) do
              yield! getAllFiles d pattern }

getAllFiles "c:\\temp" "*.xml"
|> Seq.iter (printfn "%s")
```

Factor

```
USE: io.directories.search

"." t [
    dup ".factor" tail? [ print ] [ drop ] if
] each-file
```

Forth

Works with: gforth version 0.6.2

Todo: track the full path and print it on matching files.

```
defer ls-filter

: dots? ( name len -- ? )
  dup 1 = if drop c@ [char] . =
  else 2 = if dup c@ [char] . = swap 1+ c@ [char] . = and
  else drop false then then ;

: ls-r ( dir len -- )
  open-dir if drop exit then  ( dirid)
  begin
    dup pad 256 rot read-dir throw
  while
    pad over dots? 0= if   \ ignore current and parent dirs
      pad over recurse
      pad over ls-filter if
        cr pad swap type
      else drop then
    else drop then
  repeat
  drop close-dir throw ;

: c-file? ( str len -- ? )
  dup 3 < if 2drop false exit then
  + 1- dup c@ 32 or
   dup [char] c <> swap [char] h <> and if drop false exit then
  1- dup c@ [char] . <> if drop false exit then
  drop true ;
' c-file? is ls-filter

s" ." ls-r
```

GAP

```
Walk := function(name, op)
        local dir, file, e;
        dir := Directory(name);
        for e in SortedList(DirectoryContents(name)) do
                file := Filename(dir, e);
                if IsDirectoryPath(file) then
                        if not (e in [".", ".."]) then
                                Walk(file, op);
                        fi;
                else
                        op(file);
                fi;
        od;
end;

# This will print filenames
Walk(".", Display);
```

Go

```
package main

import (
    "fmt"
    "os"
    "path/filepath"
)

func VisitFile(fp string, fi os.FileInfo, err error) error {
    if err != nil {
        fmt.Println(err) // can't walk here,
        return nil       // but continue walking elsewhere
    }
    if !!fi.IsDir() {
        return nil // not a file.  ignore.
    }
    matched, err := filepath.Match("*.mp3", fi.Name())
    if err != nil {
        fmt.Println(err) // malformed pattern
        return err       // this is fatal.
    }
    if matched {
        fmt.Println(fp)
    }
    return nil
}

func main() {
    filepath.Walk("/", VisitFile)
}
```

Groovy

Print all text files in the current directory tree

```
new File('.').eachFileRecurse {
  if (it.name =~ /.*\.txt/) println it;
}
```

GUISS

Here we list all files that match the pattern m*.txt in "My Documents" and all of its subfolders:

```
Start,Find,Files and Folders,Dropdown: Look in>My Documents,
Inputbox: filename>m*.txt,Button:Search
```

Haskell

```
import System.Environment
import System.Directory
import System.FilePath.Find

search pat dir =
  find always (fileName ~~? pat) dir

main = do [pat] <- getArgs
          dir   <- getCurrentDirectory
          files <- search pat dir
          mapM_ putStrLn files
```

IDL

```
result = file_search( directory, '*.txt', count=cc )
```

This will descend down the directory/ies in the variable `"directory"` (which can be an array) returning an array of strings with the names of the files matching "*.txt" and placing the total number of matches into the variable `"cc"`

Icon and Unicon

Icon

Icon doesn't support 'stat' or 'open' of a directory; however, information can be obtained by use of the `system` function to access command line.

Unicon

```
procedure main()
every write(!getdirs("."))  # writes out all directories from the current directory down
end

procedure getdirs(s)  #: return a list of directories beneath the directory 's'
local D,d,f

if ( stat(s).mode ? ="d" ) & ( d := open(s) ) then {
      D := [s]
      while f := read(d) do
         if not ( ".." ? =f ) then          # skip . and ..
            D |||:= getdirs(s || "/" ||f)
      close(d)
      return D
      }
end
```

J

```
require 'dir'
>{."1 dirtree '*.html'
```

The verb `dirtree` returns a file listing of a directory tree as a boxed matrix with file names in the first column. The primitives `>{."1` will return the unboxed contents of the first column.

'*.html' can be replaced by another pattern, of course.

Java

Works with: Java version 1.4+

Done using no pattern. But with end string comparison which gave better results.

```
import java.io.File;

public class MainEntry {
    public static void main(String[] args) {
        walkin(new File("/home/user")); //Replace this with a suitable directory
    }

    /**
     * Recursive function to descend into the directory tree and find all the files
     * that end with ".mp3"
     * @param dir A file object defining the top directory
     **/
    public static void walkin(File dir) {
        String pattern = ".mp3";

        File listFile[] = dir.listFiles();
        if (listFile != null) {
            for (int i=0; i<listFile.length; i++) {
                if (listFile[i].isDirectory()) {
                    walkin(listFile[i]);
                } else {
                    if (listFile[i].getName().endsWith(pattern)) {
                        System.out.println(listFile[i].getPath());
                    }
                }
            }
        }
    }
}
```

Works with: Java version 7+

Luckily, `java.nio.file.Files` gives us a `walkFileTree` method that does exactly what this task calls for.

```
import java.io.IOException;
import java.nio.file.*;
import java.nio.file.attribute.BasicFileAttributes;

public class WalkTree {
        public static void main(String[] args) throws IOException {
                Path start = FileSystems.getDefault().getPath("/path/to/file");
                Files.walkFileTree(start, new SimpleFileVisitor<Path>() {
                        @Override
                        public FileVisitResult visitFile(Path file,
                                        BasicFileAttributes attrs) throws IOException {
                                if (file.toString().endsWith(".mp3")) {
                                        System.out.println(file);
                                }
                                return FileVisitResult.CONTINUE;
                        }
                });
        }
}
```

Works with: Java version 8+

```
import java.io.IOException;
import java.nio.file.*;

public class WalkTree {
        public static void main(String[] args) throws IOException {
                Path start = FileSystems.getDefault().getPath("/path/to/file");
                Files.walk(start)
                     .filter( path -> path.toFile().isFile())
                     .filter( path -> path.toString().endsWith(".mp3"))
                     .forEach( System.out::println );
        }
}
```

JavaScript

Works with: JScript

```
var fso = new ActiveXObject("Scripting.FileSystemObject");
```

```
function walkDirectoryTree(folder, folder_name, re_pattern) {
    WScript.Echo("Files in " + folder_name + " matching '" + re_pattern + "':");
    walkDirectoryFilter(folder.files, re_pattern);

    var subfolders = folder.SubFolders;
    WScript.Echo("Folders in " + folder_name + " matching '" + re_pattern + "':");
    walkDirectoryFilter(subfolders, re_pattern);

    WScript.Echo();
    var en = new Enumerator(subfolders);
    while (! en.atEnd()) {
        var subfolder = en.item();
        walkDirectoryTree(subfolder, folder_name + "/" + subfolder.name, re_pattern);
        en.moveNext();
    }
}

function walkDirectoryFilter(items, re_pattern) {
    var e = new Enumerator(items);
    while (! e.atEnd()) {
        var item = e.item();
        if (item.name.match(re_pattern))
            WScript.Echo(item.name);
        e.moveNext();
    }
}

walkDirectoryTree(dir, dir.name, '\\.txt$');
```

Lasso

```
// care only about visible files and filter out any directories
define dir -> eachVisibleFilePath() => {
        return with name in self -> eachEntry where #name -> second != io_dir_dt_dir where not(#name -> first  ->
}

// care only about visible directories and filter out any files
define dir -> eachVisibleDir() => {
        return with name in self -> eachEntry where #name -> second == io_dir_dt_dir where not(#name -> first -> b
}

// Recursively walk the directory tree and find all files and directories
// return only paths to files
define dir -> eachVisibleFilePathRecursive(-dirFilter = void) => {
        local(files = .eachVisibleFilePath)
        with dir in .eachVisibleDir
        where !#dirFilter || #dirFilter(#dir -> realPath)
        do {
                #files = tie(#files, #dir -> eachVisibleFilePathRecursive(-dirFilter = #dirFilter))
        }
        return #files
}

local(matchingfilenames = array)

with filepath in dir('/') -> eachVisibleFilePathRecursive
where #filepath -> endswith('.lasso')
let filename = #filepath -> split('/') -> last
do #matchingfilenames -> insert(#filename)

#matchingfilenames
```

-> array(myfile.lasso, test.lasso, rosetta.lasso)

Mathematica

The built-in function `FileNames` does exactly this:

- `FileNames[]` lists all files in the current working directory.
- `FileNames[form]` lists all files in the current working directory whose names match the string pattern form.
- `FileNames[{form1,form2,...}]` lists all files whose names match any of the form_i.
- `FileNames[forms,{dir1,dir2,...}]` lists files with names matching forms in any of the directories dir_i.
- `FileNames[forms,dirs,n]` includes files that are in subdirectories up to n levels down.

Examples (find all files in current directory, find all png files in root directory, find all files on the hard drive):

```
FileNames["*"]
FileNames["*.png", $RootDirectory]
FileNames["*", {"*"}, Infinity]
```

the result can be printed with Print /@ FileNames[....]

MATLAB / Octave

```
function walk_a_directory_recursively(d, pattern)
        f = dir(fullfile(d,pattern));
        for k = 1:length(f)
                printf('%s\n',fullfile(d,f(k).name));
        end;

        f = dir(d);
        n = find([f.isdir]);
        for k=n(:)'
                if any(f(k).name~='.')
                        walk_a_directory_recursively(fullfile(d,f(k).name), pattern);
                end;
        end;
end;
```

MAXScript

```
fn walkDir dir pattern =
(
    dirArr = GetDirectories (dir + "\\*")

    for d in dirArr do
    (
        join dirArr (getDirectories (d + "\\*"))
    )

    append dirArr (dir + "\\") -- Need to include the original top level directory

    for f in dirArr do
    (
        print (getFiles (f + pattern))
    )
)

walkDir "C:" "*.txt"
```

Nimrod

```
import os, re

for file in walkDirRec "/":
  if file.match re".*\.mp3":
    echo file
```

Objective-C

```
NSString *dir = NSHomeDirectory();
NSDirectoryEnumerator *de = [[NSFileManager defaultManager] enumeratorAtPath:dir];

for (NSString *file in de)
  if ([[file pathExtension] isEqualToString:@"mp3"])
    NSLog(@"%@", file);
```

OCaml

```
#!/usr/bin/env ocaml
#load "unix.cma"
#load "str.cma"
open Unix

let walk_directory_tree dir pattern =
  let select str = Str.string_match (Str.regexp pattern) str 0 in
  let rec walk acc = function
  | [] -> (acc)
  | dir::tail ->
      let contents = Array.to_list (Sys.readdir dir) in
      let contents = List.rev_map (Filename.concat dir) contents in
      let dirs, files =
        List.fold_left (fun (dirs,files) f ->
             match (stat f).st_kind with
             | S_REG -> (dirs, f::files)  (* Regular file *)
             | S_DIR -> (f::dirs, files)  (* Directory *)
             | _ -> (dirs, files)
          ) ([],[]) contents
      in
      let matched = List.filter (select) files in
      walk (matched @ acc) (dirs @ tail)
  in
  walk [] [dir]
;;

let () =
  let results = walk_directory_tree "/usr/local/lib/ocaml"  ".*\\.cma" in
  List.iter print_endline results;
;;
```

Oz

```
declare
  [Path] = {Module.link ['x-oz://system/os/Path.ozf']}
  [Regex] = {Module.link ['x-oz://contrib/regex']}

  proc {WalkDirTree Root Pattern Proc}
     proc {Walk R}
        Entries = {Path.readdir R}
        Files = {Filter Entries Path.isFile}
        MatchingFiles = {Filter Files fun {$ File} {Regex.search Pattern File} \= false end}
        Subdirs = {Filter Entries Path.isDir}
     in
        {ForAll MatchingFiles Proc}
        {ForAll Subdirs Walk}
     end
  in
     {Walk Root}
  end
in
  {WalkDirTree "." ".*\\.oz$" System.showInfo}
```

Perl

Works with: Perl version 5.x

```
use File::Find qw(find);
my $dir     = '.';
my $pattern = 'foo';
find sub {print $File::Find::name if /$pattern/}, $dir;
```

Perl 6

Uses File::Find from File-Tools (http://github.com/tadzik/perl6-File-Tools)

```
use File::Find;

.say for find(dir => '.').grep(/foo/);
```

PHP

```
function findFiles($dir = '.', $pattern = '/./'){
  $prefix = $dir . '/';
  $dir = dir($dir);
  while (false !== ($file = $dir->read())){
    if ($file === '.' || $file === '..') continue;
    $file = $prefix . $file;
    if (is_dir($file)) findFiles($file, $pattern);
    if (preg_match($pattern, $file)){
      echo $file . "\n";
    }
  }
}
findFiles('./foo', '/\.bar$/');
```

This implementation uses Perl compatible regular expressions to match the whole path of the file

PHP BFS (Breadth First Search)

```
/*
        This script performs a BFS search with recursion protection
        it is often faster to search using this method across a
        filesystem due to a few reasons:

        * filesystem is accessed in native node order
        * a recursive function is not required allowing infinate depth
        * multiple directory handles are not required
        * the file being searched for is often not that deep in the fs
        This method also leverages PHP array hashing to speed up loop
        detection while minimizing the amount of RAM used to track the
        search history.

        -Geoffrey McRae
        Released as open license for any use.
*/
if ($_SERVER['argc'] < 3) {
        printf(
                "\n" .
                "Usage: %s (path) (search) [stop]\n" .
                "       path    the path to search\n" .
                "       search  the filename to search for\n" .
                "       stop    stop when file found, default 1\n" .
                "\n"
                , $_SERVER['argv'][0]);
        exit(1);
}

$path   = $_SERVER['argv'][1];
$search = $_SERVER['argv'][2];
if ($_SERVER['argc'] > 3)
        $stop = $_SERVER['argv'][3] == 1;
else    $stop = true;

/* get the absolute path and ensure it has a trailing slash */
$path = realpath($path);
if (substr($path, -1) !== DIRECTORY_SEPARATOR)
        $path .= DIRECTORY_SEPARATOR;

$queue = array($path => 1);
$done  = array();
$index = 0;
while(!empty($queue)) {
        /* get one element from the queue */
        foreach($queue as $path => $unused) {
                unset($queue[$path]);
                $done[$path] = null;
                break;
        }
        unset($unused);

        $dh = @opendir($path);
        if (!$dh) continue;
        while(($filename = readdir($dh)) !== false) {
                /* dont recurse back up levels */
                if ($filename == '.' || $filename == '..')
                        continue;

                /* check if the filename matches the search term */
                if ($filename == $search) {
                        echo "$path$filename\n";
                        if ($stop)
                                break 2;
                }

                /* get the full path */
                $filename = $path . $filename;

                /* resolve symlinks to their real path */
                if (is_link($filename))
                        $filename = realpath($filename);

                /* queue directories for later search */
                if (is_dir($filename)) {
                        /* ensure the path has a trailing slash */
                        if (substr($filename, -1) !== DIRECTORY_SEPARATOR)
                                $filename .= DIRECTORY_SEPARATOR;

                        /* check if we have already queued this path, or have done it */
                        if (array_key_exists($filename, $queue) || array_key_exists($filename, $done))
                                continue;

                        /* queue the file */
                        $queue[$filename] = null;
                }
        }
        closedir($dh);
}
```

PicoLisp

```
(let Dir "."
   (recur (Dir)
      (for F (dir Dir)
         (let Path (pack Dir "/" F)
            (cond
               ((=T (car (info Path)))             # Is a subdirectory?
                  (recurse Path) )                 # Yes: Recurse
               ((match '`(chop "s@.l") (chop F))   # Matches 's*.l'?
                  (println Path) ) ) ) ) ) )       # Yes: Print it
```

Output:

```
"./src64/sym.l"
"./src64/subr.l"
...
```

Pop11

Built-in procedure `sys_file_match` searches directories or directory trees using shell-like patterns (three dots indicate search for subdirectory tree).

```
lvars repp, fil;
;;; create path repeater
sys_file_match('.../*.p', '', false, 0) -> repp;
;;; iterate over paths
while (repp() ->> fil) /= termin do
     ;;; print the path
     printf(fil, '%s\n');
endwhile;
```

PowerShell

In PowerShell the `Get-ChildItem` cmdlet allows for recursive filtering on file names with simple wildcards:

```
Get-ChildItem -Recurse -Include *.mp3
```

For more complex filtering criteria the result of `Get-ChildItem` can be piped into the `Where-Object` cmdlet:

```
Get-ChildItem -Recurse |
  Where-Object { $_.Name -match 'foo[0-9]' -and $_.Length -gt 5MB }
```

To perform an action on every matching file the results can be piped into the `ForEach-Object` cmdlet:

```
Get-ChildItem -Recurse |
  Where-Object { $_.Name -match 'foo[0-9]' } |
  ForEach-Object { ... }
```

Note: To include only *files* instead of directories too each of the above needs an additional`Where-Object` filter:

```
| Where-Object { !$_.PSIsContainer }
```

PureBasic

```
Procedure.s WalkRecursive(dir,path.s,Pattern.s="\.txt$")
  Static RegularExpression
  If Not RegularExpression
    RegularExpression=CreateRegularExpression(#PB_Any,Pattern)
  EndIf

  While NextDirectoryEntry(dir)
    If DirectoryEntryType(dir)=#PB_DirectoryEntry_Directory
      If DirectoryEntryName(dir)<>"." And DirectoryEntryName(dir)<>".."
        If ExamineDirectory(dir+1,path+DirectoryEntryName(dir),"")
          WalkRecursive(dir+1,path+DirectoryEntryName(dir)+"\",Pattern)
          FinishDirectory(dir+1)
        Else
          Debug "Error in "+path+DirectoryEntryName(dir)
        EndIf
      EndIf
    Else ; e.g. #PB_DirectoryEntry_File
      If MatchRegularExpression(RegularExpression,DirectoryEntryName(dir))
        Debug DirectoryEntryName(dir)
      EndIf
    EndIf
  Wend
EndProcedure
```

```
;- Implementation; Find all .log-files in the C:\Windows tree
ExamineDirectory(1,"C:\WINDOWS\","")
WalkRecursive(1,"C:\WINDOWS\","\.log$")
FinishDirectory(1)
```

Python

Works with: Python version 3.x
Works with: Python version 2.3+

This uses the standard os.walk() (http://docs.python.org/py3k/library/os.html?highlight=os.walk#os.walk) module function to walk a directory tree, and the fnmatch (http://docs.python.org/py3k/library/fnmatch.html) module for matching file names.

```
import fnmatch
import os

rootPath = '/'
pattern = '*.mp3'

for root, dirs, files in os.walk(rootPath):
    for filename in fnmatch.filter(files, pattern):
        print( os.path.join(root, filename))
```

Works with: Python version 2.x
Works with: Python version 3.x

A more strictly comparable port of this 2.3+ code to earlier versions of Python would be:

```
from fnmatch import fnmatch
import os, os.path

def print_fnmatches(pattern, dir, files):
    for filename in files:
        if fnmatch(filename, pattern):
            print os.path.join(dir, filename)

os.path.walk('/', print_fnmatches, '*.mp3')
```

The old *os.path.walk* function was a challenge for many to use because of the need to pass a function into the walk, and any arguments to that function through to it ... as shown. It's sometimes useful to pass mutable objects (lists, dictionaries, or instances of user-defined classes) to the inner function ... for example, to collect all the matching files for later processing.

Of course the function being passed down through *os.path.walk()* can also be an instance of an object which maintains it's own data collections. Any matching criteria can be set as attributes of that object in advance and methods of that object can be called upon for later processing as well. That would the an object oriented approach which would obviate the need for the "arguments" to be passed through *os.path.walk()* at all.

Works with: Python version 2.5
Library: Path

(*Note:* This uses a non-standard replacement to the **os.path** module)

```
from path import path

rootPath = '/'
pattern = '*.mp3'

d = path(rootPath)
for f in d.walkfiles(pattern):
  print f
```

R

```
dir("/bar/foo", "mp3",recursive=T)
```

Racket

```

-> (for ([f (in-directory "/tmp")] #:when (regexp-match? "\\.rkt$" f))
     (displayln f))
... *.rkt files including in nested directories ...

```

Rascal

```
import IO;
public void Walk(loc a, str pattern){
        for (entry <- listEntries(a))
                if (endsWith(entry, pattern))
                        println(entry);
                elseif (isDirectory(a+entry))
                        Walk2(a+entry, pattern);
}
```

REALbasic

```
Sub printFiles(parentDir As FolderItem, pattern As String)
  For i As Integer = 1 To parentDir.Count
    If parentDir.Item(i).Directory Then
      printFiles(parentDir.Item(i), pattern)
    Else
      Dim rg as New RegEx
      Dim myMatch as RegExMatch
      rg.SearchPattern = pattern
      myMatch = rg.search(parentDir.Item(i).Name)
      If myMatch <> Nil Then Print(parentDir.Item(i).AbsolutePath)
    End If
  Next
End Sub
```

Accepts a FolderItem object and a Regex pattern as a string:

```

  Dim f As FolderItem = GetFolderItem("C:\Windows\system32")
  Dim pattern As String = "((?:[a-z][a-z]+))(\.)(dll)"  //all file names ending in .dll
  printFiles(f, pattern)
```

REXX

Works with: Regina

The following program was tested in a DOS window under Windows/XP and should work for all Microsoft Windows.

```
/*REXX program shows files in a  single directory that match a criteria.*/
parse arg xdir;  if xdir='' then xdir='\'         /*Any DIR? Use default.*/
@.=0                                    /*default in case ADDRESS fails. */
trace off                               /*suppress REXX err msg for fails*/
address system 'DIR' xdir '/b' with output stem @.  /*issue the DIR cmd.*/
if rc\==0  then do                                 /*an error happened?*/
                say '***error!*** from DIR' xDIR   /*indicate que pasa.*/
                say 'return code='  rc             /*show the Ret Code.*/
                exit rc                            /*exit with the  RC.*/
                end                                /* [↑]  bad address.*/
#=@.rc                                             /*number of entries.*/
if #==0  then #='   no    '                        /*use a word, ¬zero.*/
say center('directory ' xdir " has "    #      ' matching entries.',79,'—')

       do j=1  for #;  say @.j;  end    /*show files that met criteria.  */

exit @.0+rc                             /*stick a fork in it, we're done.*/
```

Ruby

Using the Find core Module:

```
require 'find'

Find.find('/your/path') do |f|
   # print file and path to screen if filename ends in ".mp3"
   puts f if f.match(/\.mp3\Z/)
end
```

A little less verbose example using a shortcut for the glob method of Dir:

```
puts Dir['**/*.mp3']
```

Scala

This is not implemented in the Scala library. Here is a simple solution, building on Java class *java.io.File*:

```
import java.io.File

object `package` {
  def walkTree(file: File): Iterable[File] = {
    val children = new Iterable[File] {
      def iterator = if (file.isDirectory) file.listFiles.iterator else Iterator.empty
    }
    Seq(file) ++: children.flatMap(walkTree(_))
  }
}

object Test extends App {
  val dir = new File("/home/user")
  for(f <- walkTree(dir)) println(f)
  for(f <- walkTree(dir) if f.getName.endsWith(".mp3")) println(f)
}
```

Scheme

Varies slightly depending on the implementation of scheme.

Works with: Chicken Scheme

```
(use posix)
(use files)
(use srfi-13)

(define (walk FN PATH)
  (for-each (lambda (ENTRY)
    (cond ((not (null? ENTRY))

           (let ((MYPATH (make-pathname PATH ENTRY)))

             (cond ((directory-exists? MYPATH)
                    (walk FN MYPATH) ))

             (FN MYPATH) )))) (directory PATH #t) ))

(walk (lambda (X) (cond ((string-suffix? ".scm" X) (display X)(newline) ))) "/home/user/")
```

See also: **(find-files ...)** function in the **posix** module.

Works with: Gauche

```
(use file.util)
(use srfi-13)

(define (walk FN PATH)
  (for-each (lambda (ENTRY)
    (cond ((not (null? ENTRY))
           (let ((MYPATH ENTRY))

             (cond ((file-is-directory? MYPATH)
                    (walk FN MYPATH) ))

             (FN MYPATH) )))) (directory-list PATH :add-path? #t :children? #t ) ))

(walk (lambda (X) (cond ((string-suffix? ".scm" X) (display X)(newline) ))) "/home/user/")
```

See also: **(find-file-in-paths ...)** function in the **file.util** module.

Works with: PLT Scheme

```
#lang scheme

(require srfi/13)

(define (walk FN PATH)
  (for-each (lambda (ENTRY)
    (cond ((not (null? ENTRY))

           (let ((MYPATH (build-path PATH ENTRY)))

             (cond ((directory-exists? MYPATH)
                    (walk FN MYPATH) ))

             (FN MYPATH) )))) (directory-list PATH)))

(walk (lambda (X) (cond ((string-suffix? ".scm" (path->string X)) (display X)(newline) ))) "/home/user/")
```

See also: **(find-files ...)** function in the **file** module.

Sample output:

```
/home/user/one.scm
/home/user/lang/two.scm
[...]
```

Seed7

Seed7 has a standard path representation (http://seed7.sourceforge.net/manual/os.htm#Standard_path_representation) , which is independent of the operating system. The function readDir (http://seed7.sourceforge.net/manual/os.htm#readDir) reads the contents of a directory as array of strings. The files . and .. are left out, so it is not necessary to ignore them. The function fileType (http://seed7.sourceforge.net/manual/os.htm#fileType) is used to determine, if a file is a directory. The example below follows symbolic links. To ignore symbolic links use fileTypeSL (http://seed7.sourceforge.net/libraries/osfiles.htm#fileTypeSL%28in_string%29) instead of fileType (http://seed7.sourceforge.net/libraries/osfiles.htm#fileType%28in_string%29) .

```
$ include "seed7_05.s7i";
  include "osfiles.s7i";

const proc: walkDir (in string: dirName, in string: extension) is func
  local
    var string: fileName is "";
    var string: path is "";
  begin
    for fileName range readDir(dirName) do
      path := dirName & "/" & fileName;
      if endsWith(path, extension) then
        writeln(path);
      end if;
      if fileType(path) = FILE_DIR then
        walkDir(path, extension);
      end if;
    end for;
  end func;

const proc: main is func
  begin
    walkDir(".", ".sd7");
  end func;
```

Smalltalk

Works with: GNU Smalltalk

```
Directory extend [
  wholeContent: aPattern do: twoBlock [
    self wholeContent: aPattern withLevel: 0 do: twoBlock.
  ]
  wholeContent: aPattern withLevel: l do: twoBlock [
    |cont|
    cont := (self contents) asSortedCollection.
    cont remove: '.'; remove: '..'.
    cont
    do: [ :n | |fn ps|
      ps := (Directory pathSeparator) asString.
      fn := (self name), ps, n.
      ((File name: fn) isDirectory)
      ifTrue: [
        twoBlock value: (n, ps) value: l.
        (Directory name: fn) wholeContent: aPattern withLevel: (l+1) do: twoBlock.
      ]
      ifFalse: [
        ( n =~ aPattern )
        ifMatched: [ :m |
          twoBlock value: n value: l
        ]
      ]
    ]
  ]
].
```

```
|d|
d := Directory name: '.'.
d wholeContent: '\.st$' do: [ :f :l |
   0 to: l do: [ :i | (Character tab) display ].
   f displayNl
].
```

Tcl

Works with: Tcl version 8.5

```
package require fileutil
proc walkin {path cmd} {
    set normalized [::fileutil::fullnormalize $path]
    set myname [lindex [info level 0] 0]
    set children [glob -nocomplain -directory $path -types hidden *]
    lappend children {*}[glob -nocomplain -directory $path *]
    foreach child $children[set children {}] {
        if {[file tail $child] in {. ..}} {
            continue
        }
        if {[file isdirectory $child]} {
            if {[file type $child] eq "link"} {
                set normalizedchild [fileutil::fullnormalize $child]
                if {[string first $normalized/ $normalizedchild] == 0} {
                    #symlink to a directory in $path.  Avoid cyclic traversal.
                    #Don't descend.
                } else {
                    $myname $child $cmd
                }
            }
        }
        {*}$cmd $child
    }
}

walkin /home/usr {apply {fname {
    set tail [file tail $fname]
    if {[string match *.mp3 $tail]} {
        puts $fname
    }
}}}
```

Visual Basic .NET

Works with: Visual Basic .NET version 9.0+

This uses the OS pattern matching

```
Sub walkTree(ByVal directory As IO.DirectoryInfo, ByVal pattern As String)
    For Each file In directory.GetFiles(pattern)
        Console.WriteLine(file.FullName)
    Next
    For Each subDir In directory.GetDirectories
        walkTree(subDir, pattern)
    Next
End Sub
```

UNIX Shell

Works with: Bourne Again SHell

The "find" command gives a one-line solution for simple patterns:

```
find . -name '*.txt' -type f
```

"find" can also be used to find files matching more complex patterns as illustrated in the section on Unix Pipes below.

Using "bash" version 4 or later, you can use "globstar" or "dotglob", depending on whether you want hidden directories to be searched:

```
#! /bin/bash
# Warning: globstar excludes hidden directories.
# Turn on recursive globbing (in this script) or exit if the option is not supported:
shopt -s globstar || exit

for f in **
do
  if [[ "$f" =~ \.txt$ ]] ; then
    echo "$f"
  fi
done
```

Here is a solution that does not use "find".

```
#! /bin/bash

indent_print()
{
    for((i=0; i < $1; i++)); do
        echo -ne "\t"
    done
    echo "$2"
}

walk_tree()
{
    local oldifs bn lev pr pmat
    if [[ $# -lt 3 ]]; then
        if [[ $# -lt 2 ]]; then
            pmat=".*"
        else
            pmat="$2"
        fi
        walk_tree "$1" "$pmat" 0
        return
    fi
    lev=$3
    [ -d "$1" ] || return
    oldifs=$IFS
    IFS="
"
    for el in $1/*; do
        bn=$(basename "$el")
        if [[ -d "$el" ]]; then
            indent_print $lev "$bn/"
            pr=$( walk_tree "$el" "$2" $(( lev + 1)) )
            echo "$pr"
        else
            if [[ "$bn" =~ $2 ]]; then
                indent_print $lev "$bn"
            fi
        fi
    done
    IFS=$oldifs
}

walk_tree "$1" "\.sh$"
```

A simplified version that gives the same output:

```
#! /usr/bin/env bash

walk_tree() {
        ls "$1" | while IFS= read i; do
```

```
                if [ -d "$1/$i" ]; then
                        echo "$i/"
                        walk_tree "$1/$i" "$2" | sed -r 's/^/\t/'
                else
                        echo "$i" | grep -E "$2"
                fi
        done
}

walk_tree "$1" "\.sh$"
```

UnixPipes

As illustrated above, the "find" command can be used with the -name option to match simple patterns. To find files matching more complex patterns, the results of "find" can be piped, e.g.

```
find . -type f | egrep '\.txt$|\.TXT$'
```

One way to run a command against each file that is found is to use "xargs", but if there is any possibility that a filename contains a space or tab character, then the following model should be used:

```
 find . -type f -name "*.txt" -print0 | xargs -0 fgrep sometext
```

zkl

```
d:=File.globular("..","s*.zkl")
```

Lots of options, here I'm using the defaults: recurse, just file matches (not directory names) and return a bit bucket of ASCIIZ strings.

Output:

```
d.pump(Console.println)
../Tmp/sieve.zkl
../Tmp/sock2.zkl
../Tmp/strands.zkl
../Tmp/sd.zkl
../Src/startup.zkl
../Src/ZenKinetic/sieve.zkl
../Tests/subscript.zkl
../Tests/Object/socket.zkl
../Tests/Object/string.zkl
```

globular will write to a object that has a write method or just call a method or function, which is nice for sending data to other threads (eg multi-threaded find/grep). To do the above example in one shot (without saving the results):

```
File.globular("..","s*.zkl",True,0,Console.println)
```

Zsh

Zsh has recursive globbing. The GLOB_DOTS option allows files beginning with a period to be matched.

```
setopt GLOB_DOTS
print -l -- **/*.txt
```

GLOB_DOTS can be set temporarily with the 'D' modifier.

```
print -l -- **/*.txt(D)
```

Retrieved from "http://rosettacode.org/mw/index.php?title=Walk_a_directory/Recursively&oldid=185498"

Categories: Programming Tasks | File System Operations | Recursion | Ada | ALGOL 68 | AutoHotkey | Batch File | BBC BASIC | C | POSIX | BSD libc | Win32 | C sharp | C++ | Boost | Caché ObjectScript | CoffeeScript | Common Lisp | CL-FAD | Clojure | D | E | Erlang | F Sharp | Factor | Forth | GAP | Go | Groovy | GUISS | Haskell | IDL | Icon | Unicon | J | Java | JavaScript | Lasso | Mathematica | MATLAB | Octave | MAXScript | Nimrod | Objective-C | OCaml | Oz | Perl | Perl 6 | PHP | PicoLisp | Pop11 | PowerShell | PureBasic | Python | Path | R | Racket | Rascal | REALbasic | REXX | Ruby | Scala | Scheme | Seed7 | Smalltalk | Tcl | Visual Basic .NET | UNIX Shell | UnixPipes | Zkl | Zsh | ACL2/Omit | Befunge/Omit | M4/Omit | MUMPS/Omit | PARI/GP/Omit | Retro/Omit | TI-89 BASIC/Omit | Unlambda/Omit | ZX Spectrum Basic/Omit