Walk a directory/Non-recursively

From Rosetta Code

Walk a given directory and print the *names* of files matching a given pattern. (How is "pattern" defined? substring match? DOS pattern? BASH pattern? ZSH pattern? Perl regular expression?)

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

Note: Please be careful when running any code presented here.

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

Contents

Ada

Works with: GCC version 4.12

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

procedure Walk_Directory
            (Directory : in String := ".";
             Pattern   : in String := "") -- empty pattern = all file names/subdirectory names
is
   Search  : Search_Type;
   Dir_Ent : Directory_Entry_Type;
begin
   Start_Search (Search, Directory, Pattern);

   while More_Entries (Search) loop
      Get_Next_Entry (Search, Dir_Ent);
      Put_Line (Simple_Name (Dir_Ent));
   end loop;

   End_Search (Search);
end Walk_Directory;
```

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 directory = get directory(".");
FOR file index TO UPB directory DO
  STRING file = directory[file index];
  IF grep in string("[Ss]ort*.[.]a68$", file, NIL, NIL) = match THEN
    print((file, new line))
  FI
OD
```

Sample output:

```
Quick_sort.a68
Shell_sort.a68
Cocktail_Sort.a68
Selection_Sort.a68
Merge_sort.a68
Bobosort.a68
Insertion_Sort.a68
Permutation_Sort.a68
```

AppleScript

AppleScript itself has limited built-in file system access. Typically, the Mac OS Finder is used to gather such information. To list all file/folders in the root directory:

```
tell application "Finder" to return name of every item in (startup disk)
--> EXAMPLE RESULT: {"Applications", "Developer", "Library", "System", "Users"}
```

To list all pdf files in user's home directory:

```
tell application "Finder" to return name of every item in (path to documents folder from user domain) whose name er
--> EXAMPLE RESULT: {"About Stacks.pdf", "Test.pdf"}
```

The key clause is the `whose` modifier keyword. The Finder can interpret many variations, including such terms as `whose name begins with`, `whose name contains`, etc. As well as boolean combinations:

```
tell application "Finder" to return name of every item in (path to documents folder from user domain) whose name do
--> RETURNS: {"Test.pdf"}
```

The Finder also supports the `entire contents` modifier keyword, which effectively performs a recursive directory scan without recursion.

```
tell application "Finder" to return name of every item in entire contents of (path to documents folder from user do
```

AutoHotkey

Display all INI files in Windows directory.

```
Loop, %A_WinDir%\*.ini
 out .= A_LoopFileName "`n"
MsgBox,% out
```

BASIC

Works with: QuickBASIC version 7
(older versions don't have `DIR$`)

DOS wildcards are rather underpowered when compared to... well... anything else.

```
DECLARE SUB show (pattern AS STRING)

show "*.*"

SUB show (pattern AS STRING)
    DIM f AS STRING
    f = DIR$(pattern)
    DO WHILE LEN(f)
        PRINT f
        f = DIR$
    LOOP
END SUB
```

BBC BASIC

Works with: BBC BASIC for Windows

```
      directory$ = "C:\Windows\"
      pattern$ = "*.ini"
      PROClistdir(directory$ + pattern$)
      END

      DEF PROClistdir(afsp$)
      LOCAL dir%, sh%, res%
      DIM dir% LOCAL 317
      SYS "FindFirstFile", afsp$, dir% TO sh%
      IF sh% <> -1 THEN
        REPEAT
          PRINT $$(dir%+44)
          SYS "FindNextFile", sh%, dir% TO res%
        UNTIL res% = 0
        SYS "FindClose", sh%
      ENDIF
      ENDPROC
```

C

Library: POSIX
Works with: POSIX version .1-2001

In this example, the pattern is a POSIX extended regular expression.

```
#include <sys/types.h>
#include <dirent.h>
#include <regex.h>
#include <stdio.h>

enum {
    WALK_OK = 0,
    WALK_BADPATTERN,
    WALK_BADOPEN,
};

int walker(const char *dir, const char *pattern)
{
    struct dirent *entry;
    regex_t reg;
    DIR *d;

    if (regcomp(&reg, pattern, REG_EXTENDED | REG_NOSUB))
        return WALK_BADPATTERN;
    if (!(d = opendir(dir)))
        return WALK_BADOPEN;
    while (entry = readdir(d))
        if (!regexec(&reg, entry->d_name, 0, NULL, 0))
            puts(entry->d_name);
    closedir(d);
    regfree(&reg);
    return WALK_OK;
}

int main()
{
    walker(".", ".\\.c$");
    return 0;
}
```

C++

Library: boost **version** 1.50.0

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

using namespace boost::filesystem;

int main()
{
  path current_dir(".");
  // list all files starting with a
  boost::regex pattern("a.*");
  for (directory_iterator iter(current_dir), end;
       iter != end;
       ++iter)
  {
    boost::smatch match;
    std::string fn = iter->path().filename().string(); // must make local variable
    if (boost::regex_match( fn, match, pattern))
    {
      std::cout << match[0] << "\n";
    }
  }
}
```

C#

```
using System;
using System.IO;

namespace DirectoryWalk
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] filePaths = Directory.GetFiles(@"c:\MyDir", "a*");
            foreach (string filename in filePaths)
                Console.WriteLine(filename);
        }
    }
}
```

ColdFusion

This example display all files and directories directly under **C:\temp** that end with *.html*

```
<cfdirectory action="list" directory="C:\temp" filter="*.html" name="dirListing">
<cfoutput query="dirListing">
  #dirListing.name# (#dirListing.type#)<br>
</cfoutput>
```

Common Lisp

```
(defun walk-directory (directory pattern)
  (directory (merge-pathnames pattern directory)))
```

Uses the filename pattern syntax provided by the CL implementation.

D

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

    dirEntries(".", "*.*", SpanMode.shallow).writeln;
}
```

E

```
def walkDirectory(directory, pattern) {
  for name => file ? (name =~ rx`.*$pattern.*`) in directory {
    println(name)
  }
}
```

Example:

```
? walkDirectory(<file:~>, "bash_")
.bash_history
.bash_profile
```

```
.bash_profile~
```

Emacs Lisp

directory-files gives filenames in a given directory, optionally restricted to those matching a regexp.

```
(directory-files "/some/dir/name"
                 nil        ;; just the filenames, not full paths
                 "\\.c\\'"  ;; regexp
                 t)         ;; don't sort the filenames
=>
("foo.c" "bar.c" ...)
```

Erlang

Use builtin function filelib:fold_files/5

Output:

```
8> filelib:fold_files( "/tmp", ".*", false, fun(File, Acc) -> [File|Acc] end, []).
["/tmp/.X0-lock","/tmp/.cron-check-4000-was-here",
 "/tmp/kerneloops.XyN0SP","/tmp/npicagwD7tf"]
9> filelib:fold_files( "/tmp", "k.*P", false, fun(File, Acc) -> [File|Acc] end, []).
["/tmp/kerneloops.XyN0SP"]
```

Euphoria

```
include file.e

procedure show(sequence pattern)
    sequence f
    f = dir(pattern)
    for i = 1 to length(f) do
        puts(1,f[i][D_NAME])
        puts(1,'\n')
    end for
end procedure

show("*.*")
```

F#

```
System.IO.Directory.GetFiles("c:\\temp", "*.xml")
|> Array.iter (printfn "%s")
```

Factor

Using unix globs. Also see the "directory." in basis/tools/files.factor.

```
USING: globs io io.directories kernel regexp sequences ;
IN: walk-directory-non-recursively
```

```
: print-files ( path pattern -- )
    [ directory-files ] [ <glob> ] bi* [ matches? ] curry filter
    [ print ] each ;
```

Ex:

```
   ( scratchpad ) "." "*.txt" print-files
   license.txt
```

Forth

Works with: gforth version 0.6.2

Gforth's directory walking functions are tied to the POSIX *dirent* functions, used by the C langauge entry above. Forth doesn't have regex support, so a simple filter function is used instead.

```
defer ls-filter ( name len -- ? )
: ls-all  2drop true ;
: ls-visible  drop c@ [char] . <> ;

: ls ( dir len -- )
  open-dir throw  ( dirid )
  begin
    dup pad 256 rot read-dir throw
  while
    pad over ls-filter if
      cr pad swap type
    else drop then
  repeat
  drop close-dir throw ;

\ only show C language source and header files (*.c *.h)
: c-file? ( str len -- ? )
  dup 3 < if 2drop false exit then
  + 1- dup c@
   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
```

Go

```
package main

import (
    "fmt"
    "path/filepath"
)

func main() {
    fmt.Println(filepath.Glob("*.go"))
}
```

Groovy

```
// *** print *.txt files in current directory
 new File('.').eachFileMatch(~/.*\.txt/) {
   println it
 }

 // *** print *.txt files in /foo/bar
 new File('/foo/bar').eachFileMatch(~/.*\.txt/) {
   println it
 }
```

Haskell

Works with: GHCi version 6.6

In this example, the pattern is a POSIX extended regular expression.

```
import System.Directory
import Text.Regex
import Data.Maybe

walk :: FilePath -> String -> IO ()
walk dir pattern = do
    filenames <- getDirectoryContents dir
    mapM_ putStrLn $ filter (isJust.(matchRegex $ mkRegex pattern)) filenames

main = walk "." ".\\.hs$"
```

HicEst

More on SYSTEM (http://www.HicEst.com/SYSTEM.htm) , OPEN (http://www.HicEst.com/OPEN.htm) , INDEX (http://www.HicEst.com/indexfnc.htm)

```
CHARACTER dirtxt='dir.txt', filename*80

SYSTEM(DIR='*.*', FIle=dirtxt) ! "file names", length, attrib, Created, LastWrite, LastAccess
OPEN(FIle=dirtxt, Format='"",', LENgth=files) ! parses column 1 ("file names")

DO nr = 1, files
  filename = dirtxt(nr,1) ! reads dirtxt row = nr, column = 1 to filename
  ! write file names with extensions "txt", or "hic", or "jpg" (case insensitive) using RegEx option =128:
  IF( INDEX(filename, "\.txt|\.hic|\.jpg", 128) ) WRITE() filename
ENDDO
```

IDL

```
f = file_search('*.txt', count=cc)
if cc gt 0 then print,f
```

(IDL is an array language - very few things are ever done in 'loops'.)

Icon and Unicon

This uses Unicon extensions for *stat* and to read directories. Icon can uses *system* to accomplish the same objective.

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

```
end

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

if ( stat(s).mode ? ="d" ) & ( d := open(s) ) then {
      while f := read(d) do
         if find(pat,f) then
            suspend f
      close(d)
      }
end
```

J

```
require 'dir'
0 dir '*.png'
0 dir '/mydir/*.txt'
```

The verb `dir` supports a number of reporting options determined by its left argument. A left argument of `0` reports just the file names.

Java

```
File dir = new File("/foo/bar");

String[] contents = dir.list();
for (String file : contents)
    if (file.endsWith(".mp3"))
        System.out.println(file);
```

JavaScript

Works with: JScript

```
var fso = new ActiveXObject("Scripting.FileSystemObject");
var dir = fso.GetFolder('test_folder');

function walkDirectory(dir, re_pattern) {
    WScript.Echo("Files in " + dir.name + " matching '" + re_pattern +"':");
    walkDirectoryFilter(dir.Files, re_pattern);

    WScript.Echo("Folders in " + dir.name + " matching '" + re_pattern +"':");
    walkDirectoryFilter(dir.Subfolders, re_pattern);
}

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();
    }
}

walkDirectory(dir, '\\.txt$');
```

Lasso

```
local(matchingfilenames = array)

dir('.') -> foreach => {#1 >> 'string' ? #matchingfilenames -> insert(#1)}

#matchingfilenames
```

-> array(mystrings.html, a_string_file.txt)

Lua

Lua itself is extremely spartanic as it is meant for embedding. Reading out a directory is not something that a minimal standard C library can do, and so minimal Lua without native extension libraries can't do it either. But lfs (LuaFileSystem) is about as standard an extension as it gets, so we use that.

```
require "lfs"
directorypath = "." -- current working directory
for filename in lfs.dir(directorypath) do
    if filename:match("%.lua$") then -- "%." is an escaped ".", "$" is end of string
        print(filename)
    end
end
```

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):

```
FileNames["*"]
FileNames["*.png", $RootDirectory]
```

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

MAXScript

```
getFiles "C:\\*.txt"
```

Nemerle

```
using System.Console;
using System.IO;

module DirWalk
```

```
{
    Main() : void
    {
        def files = Directory.GetFiles(@"C:\MyDir");                  // retrieves only files
        def files_subs = Directory.GetFileSystemEntries(@"C:\MyDir"); // also retrieves (but does not enter) sub-di
                                                                      // (like ls command)
        foreach (file in files) WriteLine(file);
    }
}
```

NetRexx

```
/* NetRexx */
options replace format comments java crossref symbols nobinary

import java.util.List

runSample(arg)
return

-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
method getFileNames(dirname, pattern) public static returns List
  dir = File(dirname)
  contents = dir.list()
  fileNames = ArrayList()
  loop fname over contents
    if fname.matches(pattern) then do
      fileNames.add(fname)
      end
    end fname
  Collections.sort(fileNames)
  return fileNames

-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
method runSample(arg) private static
  parse arg dirname pattern
  if dirname = '' then dirname = System.getProperty('user.dir')
  if pattern = '' then pattern = '^RW.*\\.nrx$'

  fileNames = getFileNames(dirname, pattern)
  say 'Search of' dirname 'for files matching pattern "'pattern'" found' fileNames.size() 'files.'
  loop fn = 0 while fn < fileNames.size()
    say (fn + 1).right(5)':' fileNames.get(fn)
    end fn

  return
```

Output:

```
Search of /Users/projects/RosettaCode/netrexx for files matching pattern "^RW.*\.nrx$" found 5 files.
    1: RWalkDir_Iter.nrx
    2: RWebScraping.nrx
    3: RWindowCreate.nrx
    4: RWriteFloatArray.nrx
    5: RWriteName3D01.nrx
```

Nimrod

```
import os

for file in walkFiles "/foo/bar/*.mp3":
  echo file
```

Objective-C

```
NSString *dir = @"/foo/bar";

// Pre-OS X 10.5
NSArray *contents = [[NSFileManager defaultManager] directoryContentsAtPath:dir];
// OS X 10.5+
NSArray *contents = [[NSFileManager defaultManager] contentsOfDirectoryAtPath:dir error:NULL];

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

Objeck

```
use IO;

bundle Default {
  class Test {
    function : Main(args : System.String[]) ~ Nil {
       dir := Directory->List("/src/code");
       for(i := 0; i < dir->Size(); i += 1;) {
         if(dir[i]->EndsWith(".obs")) {
           dir[i]->PrintLine();
        };
      };
    }
  }
}
```

OCaml

```
#load "str.cma"
let contents = Array.to_list (Sys.readdir ".") in
let select pat str = Str.string_match (Str.regexp pat) str 0 in
List.filter (select ".*\\.jpg") contents
```

Oz

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

  Files = {Filter {Path.readdir "."} Path.isFile}
  Pattern = ".*\\.oz$"
  MatchingFiles = {Filter Files fun {$ File} {Regex.search Pattern File} \= false end}
in
  {ForAll MatchingFiles System.showInfo}
```

Pascal

Works with: Free Pascal

```
{$H+}
```

```
program Walk;

uses SysUtils;

var Res: TSearchRec;
    Pattern, Path, Name: String;
    FileAttr: LongInt;
    Attr: Integer;

begin
   Write('File pattern: ');
   ReadLn(Pattern);            { For example .\*.pas }

   Attr := faAnyFile;
   if FindFirst(Pattern, Attr, Res) = 0 then
   begin
      Path := ExtractFileDir(Pattern);
      repeat
         Name := ConcatPaths([Path, Res.Name]);
         FileAttr := FileGetAttr(Name);
         if FileAttr and faDirectory = 0 then
         begin
            { Do something with file name }
            WriteLn(Name);
         end
      until FindNext(Res) <> 0;
   end;
   FindClose(Res);
end.
```

Perl

```
use 5.010;
my $pattern = qr{ \A a }xmso; # match files whose first character is 'a'
opendir my $dh, 'the_directory';
say for grep { $pattern } readdir $dh;
closedir $dh;
```

Or using globbing, with the `<>` operator,

```
use 5.010; say while </home/foo/bar/*.php>;
```

Or the same with the builtin `glob()` function,

```
my @filenames = glob('/home/foo/bar/*.php');
```

The `glob()` function takes any expression for its pattern, whereas `<>` is only for a literal.

```
my $pattern = '*.c';
my @filenames = glob($pattern);
```

Perl 6

The `dir` function takes the directory to traverse, and optionally a named parameter `test`, which can for example be a regex:

```
.say for dir(".", :test(/foo/))
```

PHP

Works with: PHP version 5.2.0

```
$pattern = 'php';
$dh = opendir('c:/foo/bar'); // Or '/home/foo/bar' for Linux
while (false !== ($file = readdir($dh)))
{
    if ($file != '.' and $file != '..')
    {
        if (preg_match("/$pattern/", $file))
        {
            echo "$file matches $pattern\n";
        }
    }
}
closedir($dh);
```

Or:

```
$pattern = 'php';
foreach (scandir('/home/foo/bar') as $file)
{
    if ($file != '.' and $file != '..')
    {
        if (preg_match("/$pattern/", $file))
        {
            echo "$file matches $pattern\n";
        }
    }
}
```

Works with: PHP version 4 >= 4.3.0 or 5

```
foreach (glob('/home/foo/bar/*.php') as $file){
    echo "$file\n";
}
```

PicoLisp

```
(for F (dir "@src/")                          # Iterate directory
   (when (match '`(chop "s@.c") (chop F))     # Matches 's*.c'?
      (println F) ) )                         # Yes: Print it
```

Output:

```
"start.c"
"ssl.c"
"subr.c"
"sym.c"
...
```

Pike

```
array(string) files = get_dir("/home/foo/bar");
foreach(files, string file)
    write(file + "\n");
```

Pop11

Built-in procedure sys_file_match searches directories (or directory trees) using shell-like patterns:

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

PowerShell

Since PowerShell is also a shell it should come as no surprise that this task is very simple. Listing the names of all text files, or the names of all files, starting with "f":

```
Get-ChildItem *.txt -Name
Get-ChildItem f* -Name
```

The `-Name` parameter tells the `Get-ChildItem` to return only the file names as string, otherwise a complete `FileInfo` or `DirectoryInfo` object would be returned, containing much more information than only the file name.

More complex matching can be accomplished by filtering the complete list of files using the `Where-Object` cmdlet. The following will output all file names that contain at least one vowel:

```
Get-ChildItem -Name | Where-Object { $_ -match '[aeiou]' }
```

PureBasic

The match is made using DOS wildcards. It could easily be modified to match based on a regular expression if desired (i.e. using the PCRE library).

```
Procedure walkDirectory(directory.s = "", pattern.s = "")
  Protected directoryID
  
  directoryID = ExamineDirectory(#PB_Any,directory,pattern)
  If directoryID
    While NextDirectoryEntry(directoryID)
      PrintN(DirectoryEntryName(directoryID))
    Wend
    FinishDirectory(directoryID)
  EndIf
EndProcedure

If OpenConsole()
  walkDirectory()
  
  Print(#CRLF$ + #CRLF$ + "Press ENTER to exit")
  Input()
  CloseConsole()
EndIf
```

Python

The glob (http://python.org/doc/lib/module-glob.html) library included with Python lists files matching shell-like patterns:

```
import glob
for filename in glob.glob('/foo/bar/*.mp3'):
    print filename
```

Or manually:

```
import os
for filename in os.listdir('/foo/bar'):
    if filename.endswith('.mp3'):
        print filename
```

R

```
dir("/foo/bar", "mp3")
```

Racket

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

Raven

```
'dir://.' open each as item
    item m/\.txt$/ if "%(item)s\n" print
```

Rascal

```
import IO;
public void Walk(loc a, str pattern){
        for (entry <- listEntries(a))
                endsWith(entry, pattern) ? println(entry);
}
```

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 directory tree that match a given 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 /s' with output stem @.   /*issue 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

```
# Files under this directory:
Dir.glob('*') { |file| puts file }

# Files under path '/foo/bar':
Dir.glob( File.join('/foo/bar', '*') ) { |file| puts file }

# As a method
def file_match(pattern=/\.txt/, path='.')
  Dir[File.join(path,'*')].each do |file|
    puts file if file =~ pattern
  end
end
```

Scala

```
import java.io.File

val dir = new File("/foo/bar").list()
dir.filter(file => file.endsWith(".mp3")).foreach(println)
```

Seed7

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

const proc: main is func
  local
    var string: fileName is "";
  begin
    for fileName range readDir(".") do
      if endsWith(fileName, ".sd7") then
        writeln(fileName);
      end if;
    end for;
  end func;
```

Smalltalk

```
(Directory name: 'a_directory')
```

```
  allFilesMatching: '*.st' do: [ :f | (f name) displayNl ]
```

Tcl

For the current directory:

```
foreach filename [glob *.txt] {
    puts $filename
}
```

For an arbitrary directory:

```
set dir /foo/bar
foreach filename [glob -directory $dir *.txt] {
    puts $filename
    ### Or, if you only want the local filename part...
    # puts [file tail $filename]
}
```

Toka

As with the C example, this uses a a POSIX extended regular expression as the pattern. The **dir.listByPattern** function used here was introduced in library revision 1.3.

```
needs shell
" ."  " .\\.txt$" dir.listByPattern
```

TUSCRIPT

```
$$ MODE TUSCRIPT
files=FILE_NAMES (+,-std-)
fileswtxt= FILTER_INDEX (files,":*.txt:",-)
txtfiles= SELECT (files,#fileswtxt)
```

Output:

```
files     DEST'MAKROS'ROSETTA.TXT'SKRIPTE'STUDENTS.XML'TUSTEP.INI
fileswtxt 3
txtfiles  ROSETTA.TXT
```

Visual Basic .NET

Works with: Visual Basic .NET version 9.0+

```
'Using the OS pattern matching
For Each file In IO.Directory.GetFiles("\temp", "*.txt")
  Console.WriteLine(file)
Next

'Using VB's pattern matching and LINQ
For Each file In (From name In IO.Directory.GetFiles("\temp") Where name Like "*.txt")
  Console.WriteLine(file)
```

```
Next

'Using VB's pattern matching and dot-notation
For Each file In IO.Directory.GetFiles("\temp").Where(Function(f) f Like "*.txt")
  Console.WriteLine(file)
Next
```

UNIX Shell

```
ls -d *.c                # *.c files in current directory
(cd mydir && ls -d *.c)  # *.c files in mydir
```

`*.c` is a *file name pattern*, also known as a *glob pattern*. The shell expands each pattern to a sorted list of matching files. Details are in your shell's manual.

If there are no *.c files, `ls` fails with an error message.

UnixPipes

Here using grep for regexp.

```
ls | grep '\.c$'
```

zkl

Unix glob, with wildcarding and options on file type, case folding and a few others.

```
File.glob("*.zkl") //--> list of matches
```

Zsh

Zsh has powerful filename generation features, which can filter by file names, permissions, size, type, etc.

```
print -l -- *.c
```

Retrieved from "http://rosettacode.org/mw/index.php?title=Walk_a_directory/Non-recursively&oldid=185497"
Categories: Programming Tasks | File System Operations | Ada | ALGOL 68 | AppleScript | AutoHotkey | BASIC | BBC BASIC | C | POSIX | C++ | Boost | C sharp | ColdFusion | Common Lisp | D | E | Emacs Lisp | Erlang | Euphoria | F Sharp | Factor | Forth | Go | Groovy | Haskell | HicEst | IDL | Icon | Unicon | J | Java | JavaScript | Lasso | Lua | Mathematica | MAXScript | Nemerle | NetRexx | Nimrod | Objective-C | Objeck | OCaml | Oz | Pascal | Perl | Perl 6 | PHP | PicoLisp | Pike | Pop11 | PowerShell | PureBasic | Python | R | Racket | Raven | Rascal | REXX | Ruby | Scala | Seed7 | Smalltalk | Tcl | Toka | TUSCRIPT | Visual Basic .NET | UNIX Shell | UnixPipes | Zkl | Zsh | AWK/Omit | Bc/Omit | Befunge/Omit | Dc/Omit | GUISS/Omit | M4/Omit | Make/Omit | PARI/GP/Omit | Retro/Omit | TI-89 BASIC/Omit

Unlambda/Omit