NAME

File::Which – Perl implementation of the which utility as an API

VERSION

version 1.23

SYNOPSIS

```
use File::Which;                    # exports which()
use File::Which qw(which where);  # exports which() and where()

my $exe_path = which 'perldoc';

my @paths = where 'perl';
# Or
my @paths = which 'perl'; # an array forces search for all of them
```

DESCRIPTION

File::Which finds the full or relative paths to executable programs on the system. This is normally the function of `which` utility. `which` is typically implemented as either a program or a built in shell command. On some platforms, such as Microsoft Windows it is not provided as part of the core operating system. This module provides a consistent API to this functionality regardless of the underlying platform.

The focus of this module is correctness and portability. As a consequence platforms where the current directory is implicitly part of the search path such as Microsoft Windows will find executables in the current directory, whereas on platforms such as UNIX where this is not the case executables in the current directory will only be found if the current directory is explicitly added to the path.

If you need a portable `which` on the command line in an environment that does not provide it, install App::pwhich which provides a command line interface to this API.

Implementations

File::Which searches the directories of the user's `PATH` (the current implementation uses File::Spec#path to determine the correct `PATH`), looking for executable files having the name specified as a parameter to ''which''. Under Win32 systems, which do not have a notion of directly executable files, but uses special extensions such as `.exe` and `.bat` to identify them, `File::Which` takes extra steps to assure that you will find the correct file (so for example, you might be searching for `perl`, it'll try *perl.exe*, *perl.bat*, etc.)

*Linux, *BSD and other UNIXes*

There should not be any surprises here. The current directory will not be searched unless it is explicitly added to the path.

Modern Windows (including NT, XP, Vista, 7, 8, 10 etc)

Windows NT has a special environment variable called `PATHEXT`, which is used by the shell to look for executable files. Usually, it will contain a list in the form `.EXE;.BAT;.COM;.JS;.VBS` etc. If `File::Which` finds such an environment variable, it parses the list and uses it as the different extensions.

Cygwin

Cygwin provides a Unix-like environment for Microsoft Windows users. In most ways it works like other Unix and Unix-like environments, but in a few key aspects it works like Windows. As with other Unix environments, the current directory is not included in the search unless it is explicitly included in the search path. Like on Windows, files with `.EXE` or <.BAT> extensions will be discovered even if they are not part of the query. `.COM` or extensions specified using the `PATHEXT` environment variable will NOT be discovered without the fully qualified name, however.

Windows ME, 98, 95, MS-DOS, OS/2

This set of operating systems don't have the `PATHEXT` variable, and usually you will find executable files there with the extensions `.exe`, `.bat` and (less likely) `.com`. `File::Which` uses this hardcoded list if it's running under Win32 but does not find a `PATHEXT` variable.

As of 2015 none of these platforms are tested frequently (or perhaps ever), but the current maintainer is determined not to intentionally remove support for older operating systems.

VMS

Same case as Windows 9x: uses `.exe` and `.com` (in that order).

As of 2015 the current maintainer does not test on VMS, and is in fact not certain it has ever been tested on VMS. If this platform is important to you and you can help me verify and or support it on that platform please contact me.

FUNCTIONS

which

```
my $path = which $short_exe_name;
my @paths = which $short_exe_name;
```

Exported by default.

`$short_exe_name` is the name used in the shell to call the program (for example, `perl`).

If it finds an executable with the name you specified, `which()` will return the absolute path leading to this executable (for example, */usr/bin/perl* or *C:\Perl\Bin\perl.exe*).

If it does *not* find the executable, it returns `undef`.

If `which()` is called in list context, it will return *all* the matches.

where

```
my @paths = where $short_exe_name;
```

Not exported by default.

Same as "which" in array context. Similar to the `where` csh built-in command or `which -a` command for platforms that support the `-a` option. Will return an array containing all the path names matching `$short_exe_name`.

GLOBALS

$IMPLICIT_CURRENT_DIR

True if the current directory is included in the search implicitly on whatever platform you are using. Normally the default is reasonable, but on Windows the current directory is included implicitly for older shells like `cmd.exe` and `command.com`, but not for newer shells like PowerShell. If you overrule this default, you should ALWAYS localize the variable to the tightest scope possible, since setting this variable from a module can affect other modules. Thus on Windows you can get the correct result if the user is running either `cmd.exe` or PowerShell on Windows you can do this:

```
use File::Which qw( which );
use Shell::Guess;

my $path = do {
  my $is_power = Shell::Guess->running_shell->is_power;
  local $File::Which::IMPLICIT_CURRENT_DIR = !$is_power;
  which 'foo';
};
```

For a variety of reasons it is difficult to accurately compute the shell that a user is using, but Shell::Guess makes a reasonable effort.

CAVEATS

This module has no non-core requirements for Perl 5.6.2 and better.

This module is fully supported back to Perl 5.8.1. It may work on 5.8.0. It should work on Perl 5.6.x and I may even test on 5.6.2. I will accept patches to maintain compatibility for such older Perls, but you may need to fix it on 5.6.x / 5.8.0 and send me a patch.

Not tested on VMS although there is platform specific code for those. Anyone who haves a second would be

very kind to send me a report of how it went.

SUPPORT

Bugs should be reported via the GitHub issue tracker

<https://github.com/plicease/File-Which/issues>

For other issues, contact the maintainer.

SEE ALSO

pwhich, App::pwhich
: Command line interface to this module.

IPC::Cmd
: This module provides (among other things) a `can_run` function, which is similar to `which`. It is a much heavier module since it does a lot more, and if you use `can_run` it pulls in ExtUtils::MakeMaker. This combination may be overkill for applications which do not need IPC::Cmd's complicated interface for running programs, or do not need the memory overhead required for installing Perl modules.

  At least some older versions will find executables in the current directory, even if the current directory is not in the search path (which is the default on modern Unix).

  `can_run` converts directory path name to the 8.3 version on Windows using `Win32::GetShortPathName` in some cases. This is frequently useful for tools that just need to run something using `system` in scalar mode, but may be inconvenient for tools like App::pwhich where user readability is a premium. Relying on `Win32::GetShortPathName` to produce filenames without spaces is problematic, as 8.3 filenames can be turned off with tweaks to the registry (see <https://technet.microsoft.com/en-us/library/cc959352.aspx>).

Devel::CheckBin
: This module purports to "check that a command is available", but does not provide any documentation on how you might use it.

AUTHORS

- Per Einar Ellefsen <pereinar@cpan.org>
- Adam Kennedy <adamk@cpan.org>
- Graham Ollis <plicease@cpan.org>

COPYRIGHT AND LICENSE