



Linux Ubuntu 22.4.5 Manual Pages on command 'File::Find::Rule.3pm'

\$ man File::Find::Rule.3pm

File::Find::Rule(3pm) User Contributed Perl Documentation File::Find::Rule(3pm)

NAME

File::Find::Rule - Alternative interface to File::Find

SYNOPSIS

```
use File::Find::Rule;

# find all the subdirectories of a given directory
my @subdirs = File::Find::Rule->directory->in( $directory );

# find all the .pm files in @INC
my @files = File::Find::Rule->file()
    ->name( '*.pm' )
    ->in( @INC );

# as above, but without method chaining
my $rule = File::Find::Rule->new;
$rule->file;
$rule->name( '*.pm' );
my @files = $rule->in( @INC );
```

DESCRIPTION

File::Find::Rule is a friendlier interface to File::Find. It allows you to build rules which specify the desired files and directories.

METHODS

"new"

A constructor. You need not invoke "new" manually unless you wish to, as each

of the rule-making methods will auto-create a suitable object if called as class methods.

Matching Rules

"name(@patterns)"

Specifies names that should match. May be globs or regular expressions.

```
$set->name( '*.mp3', '*.ogg' ); # mp3s or oggs
```

```
$set->name( qr/^(mp3|ogg)$/ ); # the same as a regex
```

```
$set->name( 'foo.bar' ); # just things named foo.bar
```

-X tests

Synonyms are provided for each of the -X tests. See "-X" in `perlfunc` for details. None of these methods take arguments.

Test Method	Test Method
----- -----	----- -----
-r readable	-R r_readable
-w writeable	-W r_writeable
-w writable	-W r_writable
-x executable	-X r_executable
-o owned	-O r_owned
-e exists	-f file
-z empty	-d directory
-s nonempty	-l symlink
	-p fifo
-u setuid	-S socket
-g setgid	-b block
-k sticky	-c character
	-t tty
-M modified	
-A accessed	-T ascii
-C changed	-B binary

Though some tests are fairly meaningless as binary flags ("modified", "accessed", "changed"), they have been included for completeness.

```
# find nonempty files
```

```
$rule->file,  
->nonempty;
```

stat tests

The following "stat" based methods are provided: "dev", "ino", "mode", "nlink", "uid", "gid", "rdev", "size", "atime", "mtime", "ctime", "blksize", and "blocks". See "stat" in perlfunc for details.

Each of these can take a number of targets, which will follow Number::Compare semantics.

```
$rule->size( 7 );      # exactly 7  
$rule->size( ">7Ki" ); # larger than 7 * 1024 * 1024 bytes  
$rule->size( ">=7" )  
->size( "<=90" );    # between 7 and 90, inclusive  
$rule->size( 7, 9, 42 ); # 7, 9 or 42
```

"any(@rules)"

"or(@rules)"

Allows shortcircuiting boolean evaluation as an alternative to the default and-like nature of combined rules. "any" and "or" are interchangeable.

```
# find avis, movs, things over 200M and empty files
```

```
$rule->any( File::Find::Rule->name( '*.avi', '*.mov' ),  
          File::Find::Rule->size( '>200M' ),  
          File::Find::Rule->file->empty,  
          );
```

"none(@rules)"

"not(@rules)"

Negates a rule. (The inverse of "any".) "none" and "not" are interchangeable.

```
# files that aren't 8.3 safe
```

```
$rule->file  
->not( $rule->new->name( qr/^[^.]{{1,8}}(\.[^.]{{0,3}})?$/ ) );
```

"prune"

Traverse no further. This rule always matches.

"discard"

Don't keep this file. This rule always matches.

"exec(\&subroutine(\$shortname, \$path, \$fullname))"

Allows user-defined rules. Your subroutine will be invoked with `$_` set to the current short name, and with parameters of the name, the path you're in, and the full relative filename.

Return a true value if your rule matched.

```
# get things with long names
```

```
$rules->exec( sub { length > 20 } );
```

"grep(@specifiers)"

Opens a file and tests it each line at a time.

For each line it evaluates each of the specifiers, stopping at the first successful match. A specifier may be a regular expression or a subroutine.

The subroutine will be invoked with the same parameters as an `->exec` subroutine.

It is possible to provide a set of negative specifiers by enclosing them in anonymous arrays. Should a negative specifier match the iteration is aborted and the clause is failed. For example:

```
$rule->grep( qr/^\#!.*\bperl/, [ sub { 1 } ] );
```

Is a passing clause if the first line of a file looks like a perl shebang line.

"maxdepth(\$level)"

Descend at most `$level` (a non-negative integer) levels of directories below the starting point.

May be invoked many times per rule, but only the most recent value is used.

"mindepth(\$level)"

Do not apply any tests at levels less than `$level` (a non-negative integer).

"extras(\%extras)"

Specifies extra values to pass through to "File::File::find" as part of the options hash.

For example this allows you to specify following of symlinks like so:

```
my $rule = File::Find::Rule->extras({ follow => 1 });
```

May be invoked many times per rule, but only the most recent value is used.

"relative"

Trim the leading portion of any path found

"canonpath"

Normalize paths found using "File::Spec->canonpath". This will return paths

with a file-seperator that is native to your OS (as determined by File::Spec), instead of the default "/".

For example, this will return "tmp/foobar" on Unix-ish OSes and "tmp\foobar" on Win32.

"not_**"

Negated version of the rule. An effective shorthand related to ! in the procedural interface.

```
$foo->not_name('*.pl');
```

```
$foo->not( $foo->new->name('*.pl' ) );
```

Query Methods

"in(@directories)"

Evaluates the rule, returns a list of paths to matching files and directories.

"start(@directories)"

Starts a find across the specified directories. Matching items may then be queried using "match". This allows you to use a rule as an iterator.

```
my $rule = File::Find::Rule->file->name("*.jpeg")->start( "/web" );
```

```
while ( defined ( my $image = $rule->match ) ) {
```

```
    ...
```

```
}
```

"match"

Returns the next file which matches, false if there are no more.

Extensions

Extension modules are available from CPAN in the File::Find::Rule namespace. In

order to use these extensions either use them directly:

```
use File::Find::Rule::ImageSize;
```

```
use File::Find::Rule::MMagic;
```

```
# now your rules can use the clauses supplied by the ImageSize and
```

```
# MMagic extension
```

or, specify that File::Find::Rule should load them for you:

```
use File::Find::Rule qw( :ImageSize :MMagic );
```

For notes on implementing your own extensions, consult File::Find::Rule::Extending

Further examples

Finding perl scripts

```

my $finder = File::Find::Rule->or
(
  File::Find::Rule->name( '*.pl' ),
  File::Find::Rule->exec(
    sub {
      if (open my $fh, $_) {
        my $shebang = <$fh>;
        close $fh;
        return $shebang =~ /^#!.*\bperl/;
      }
      return 0;
    }
  ),
);

```

Based upon this message <http://use.perl.org/comments.pl?sid=7052&cid=10842>

ignore CVS directories

```

my $rule = File::Find::Rule->new;
$rule->or($rule->new
  ->directory
  ->name('CVS')
  ->prune
  ->discard,
  $rule->new);

```

Note here the use of a null rule. Null rules match anything they see, so the effect is to match (and discard) directories called 'CVS' or to match anything.

TWO FOR THE PRICE OF ONE

File::Find::Rule also gives you a procedural interface. This is documented in

File::Find::Rule::Procedural

EXPORTS

"find", "rule"

TAINT MODE INTERACTION

As of 0.32 File::Find::Rule doesn't capture the current working directory in a taint-unsafe manner. File::Find itself still does operations that the taint system will flag as insecure but you can use the "extras" feature to ask File::Find to

internally "untaint" file paths with a regex like so:

```
my $rule = File::Find::Rule->extras({ untaint => 1 });
```

Please consult File::Find's documentation for "untaint", "untaint_pattern", and "untaint_skip" for more information.

BUGS

The code makes use of the "our" keyword and as such requires perl version 5.6.0 or newer.

Currently it isn't possible to remove a clause from a rule object. If this becomes a significant issue it will be addressed.

AUTHOR

Richard Clamp <richardc@unixbeard.net> with input gained from this [use.perl](http://use.perl.org/~richardc/journal/6467) discussion: <http://use.perl.org/~richardc/journal/6467>

Additional proofreading and input provided by Kake, Greg McCarroll, and Andy Lester andy@petdance.com.

COPYRIGHT

Copyright (C) 2002, 2003, 2004, 2006, 2009, 2011 Richard Clamp. All Rights Reserved.

This module is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

SEE ALSO

[File::Find](#), [Text::Glob](#), [Number::Compare](#), [find\(1\)](#)

If you want to know about the procedural interface, see

[File::Find::Rule::Procedural](#), and if you have an idea for a neat extension

[File::Find::Rule::Extending](#)

perl v5.20.2

2015-12-06

[File::Find::Rule\(3pm\)](#)