NAME

File::Fetch - A generic file fetching mechanism

SYNOPSIS

```
use File::Fetch;

### build a File::Fetch object ###
my $ff = File::Fetch->new(uri => 'http://some.where.com/dir/a.txt');

### fetch the uri to cwd() ###
my $where = $ff->fetch() or die $ff->error;

### fetch the uri to /tmp ###
my $where = $ff->fetch( to => '/tmp' );

### parsed bits from the uri ###
$ff->uri;
$ff->scheme;
$ff->host;
$ff->path;
$ff->file;
```

DESCRIPTION

File::Fetch is a generic file fetching mechanism.

It allows you to fetch any file pointed to by a `ftp`, `http`, `file`, or `rsync` uri by a number of different means.

See the `HOW IT WORKS` section further down for details.

ACCESSORS

A `File::Fetch` object has the following accessors

$ff->uri

The uri you passed to the constructor

$ff->scheme

The scheme from the uri (like 'file', 'http', etc)

$ff->host

The hostname in the uri. Will be empty if host was originally 'localhost' for a 'file://' url.

$ff->vol

On operating systems with the concept of a volume the second element of a file:// is considered to the be volume specification for the file. Thus on Win32 this routine returns the volume, on other operating systems this returns nothing.

On Windows this value may be empty if the uri is to a network share, in which case the 'share' property will be defined. Additionally, volume specifications that use '|' as ':' will be converted on read to use ':'.

On VMS, which has a volume concept, this field will be empty because VMS file specifications are converted to absolute UNIX format and the volume information is transparently included.

$ff->share

On systems with the concept of a network share (currently only Windows) returns the

sharename from a file:/// url. On other operating systems returns empty.

$ff->path

The path from the uri, will be at least a single '/'.

$ff->file

The name of the remote file. For the local file name, the result of $ff->output_file will be used.

$ff->output_file

The name of the output file. This is the same as $ff->file, but any query parameters are stripped off. For example:

```
http://example.com/index.html?x=y
```

would make the output file be `index.html` rather than `index.html?x=y`.

METHODS

$ff = File::Fetch->new(uri => 'http://some.where.com/dir/file.txt');

Parses the uri and creates a corresponding File::Fetch::Item object, that is ready to be `fetch`ed and returns it.

Returns false on failure.

$ff->fetch([to => /my/output/dir/])

Fetches the file you requested. By default it writes to `cwd()`, but you can override that by specifying the `to` argument.

Returns the full path to the downloaded file on success, and false on failure.

$ff->error([BOOL])

Returns the last encountered error as string. Pass it a true value to get the `Carp::longmess()` output instead.

HOW IT WORKS

File::Fetch is able to fetch a variety of uris, by using several external programs and modules.

Below is a mapping of what utilities will be used in what order for what schemes, if available:

```
file    => LWP, file
http    => LWP, wget, curl, lynx
ftp     => LWP, Net::FTP, wget, curl, ncftp, ftp
rsync   => rsync
```

If you'd like to disable the use of one or more of these utilities and/or modules, see the `$BLACKLIST` variable further down.

If a utility or module isn't available, it will be marked in a cache (see the `$METHOD_FAIL` variable further down), so it will not be tried again. The `fetch` method will only fail when all options are exhausted, and it was not able to retrieve the file.

A special note about fetching files from an ftp uri:

By default, all ftp connections are done in passive mode. To change that, see the `$FTP_PASSIVE` variable further down.

Furthermore, ftp uris only support anonymous connections, so no named user/password pair can be passed along.

`/bin/ftp` is blacklisted by default; see the `$BLACKLIST` variable further down.

GLOBAL VARIABLES

The behaviour of File::Fetch can be altered by changing the following global variables:

$File::Fetch::FROM_EMAIL

This is the email address that will be sent as your anonymous ftp password.

Default is `File-Fetch@example.com`.

$File::Fetch::USER_AGENT

This is the useragent as `LWP` will report it.

Default is `File::Fetch/$VERSION`.

$File::Fetch::FTP_PASSIVE

This variable controls whether the environment variable `FTP_PASSIVE` and any passive switches to commandline tools will be set to true.

Default value is 1.

Note: When $FTP_PASSIVE is true, `ncftp` will not be used to fetch files, since passive mode can only be set interactively for this binary

$File::Fetch::TIMEOUT

When set, controls the network timeout (counted in seconds).

Default value is 0.

$File::Fetch::WARN

This variable controls whether errors encountered internally by `File::Fetch` should be `carp`'d or not.

Set to false to silence warnings. Inspect the output of the `error()` method manually to see what went wrong.

Defaults to `true`.

$File::Fetch::DEBUG

This enables debugging output when calling commandline utilities to fetch files. This also enables `Carp::longmess` errors, instead of the regular `carp` errors.

Good for tracking down why things don't work with your particular setup.

Default is 0.

$File::Fetch::BLACKLIST

This is an array ref holding blacklisted modules/utilities for fetching files with.

To disallow the use of, for example, `LWP` and `Net::FTP`, you could set $File::Fetch::BLACKLIST to:

```
$File::Fetch::BLACKLIST = [qw|lwp netftp|]
```

The default blacklist is [qw|ftp|], as `/bin/ftp` is rather unreliable.

See the note on `MAPPING` below.

$File::Fetch::METHOD_FAIL

This is a hashref registering what modules/utilities were known to fail for fetching files (mostly because they weren't installed).

You can reset this cache by assigning an empty hashref to it, or individually remove keys.

See the note on MAPPING below.

MAPPING

Here's a quick mapping for the utilities/modules, and their names for the $BLACKLIST, $METHOD_FAIL and other internal functions.

```
LWP         => lwp
Net::FTP    => netftp
wget        => wget
lynx        => lynx
ncftp       => ncftp
ftp         => ftp
curl        => curl
rsync       => rsync
```

FREQUENTLY ASKED QUESTIONS

So how do I use a proxy with File::Fetch?

File::Fetch currently only supports proxies with LWP::UserAgent. You will need to set your environment variables accordingly. For example, to use an ftp proxy:

```
$ENV{ftp_proxy} = 'foo.com';
```

Refer to the LWP::UserAgent manpage for more details.

I used 'lynx' to fetch a file, but its contents is all wrong!

lynx can only fetch remote files by dumping its contents to STDOUT, which we in turn capture. If that content is a 'custom' error file (like, say, a 404 handler), you will get that contents instead.

Sadly, lynx doesn't support any options to return a different exit code on non-200 OK status, giving us no way to tell the difference between a 'successfull' fetch and a custom error page.

Therefor, we recommend to only use lynx as a last resort. This is why it is at the back of our list of methods to try as well.

Files I'm trying to fetch have reserved characters or non-ASCII characters in them. What do I do?

File::Fetch is relatively smart about things. When trying to write a file to disk, it removes the query parameters (see the output_file method for details) from the file name before creating it. In most cases this suffices.

If you have any other characters you need to escape, please install the URI::Escape module from CPAN, and pre-encode your URI before passing it to File::Fetch. You can read about the details of URIs and URI encoding here:

```
http://www.faqs.org/rfcs/rfc2396.html
```

TODO

Implement $PREFER_BIN

To indicate to rather use commandline tools than modules

BUG REPORTS

Please report bugs or other issues to <bug-file-fetch@rt.cpan.org<gt>.

AUTHOR

This module by Jos Boumans <kane@cpan.org>.

COPYRIGHT