NAME

Net::FTP - FTP Client class

SYNOPSIS

```
use Net::FTP;

$ftp = Net::FTP->new("some.host.name", Debug => 0)
  or die "Cannot connect to some.host.name: $@";

$ftp->login("anonymous",'-anonymous@')
  or die "Cannot login ", $ftp->message;

$ftp->cwd("/pub")
  or die "Cannot change working directory ", $ftp->message;

$ftp->get("that.file")
  or die "get failed ", $ftp->message;

$ftp->quit;
```

DESCRIPTION

`Net::FTP` is a class implementing a simple FTP client in Perl as described in RFC959. It provides wrappers for a subset of the RFC959 commands.

OVERVIEW

FTP stands for File Transfer Protocol. It is a way of transferring files between networked machines. The protocol defines a client (whose commands are provided by this module) and a server (not implemented in this module). Communication is always initiated by the client, and the server responds with a message and a status code (and sometimes with data).

The FTP protocol allows files to be sent to or fetched from the server. Each transfer involves a **local file** (on the client) and a **remote file** (on the server). In this module, the same file name will be used for both local and remote if only one is specified. This means that transferring remote file `/path/to/file` will try to put that file in `/path/to/file` locally, unless you specify a local file name.

The protocol also defines several standard **translations** which the file can undergo during transfer. These are ASCII, EBCDIC, binary, and byte. ASCII is the default type, and indicates that the sender of files will translate the ends of lines to a standard representation which the receiver will then translate back into their local representation. EBCDIC indicates the file being transferred is in EBCDIC format. Binary (also known as image) format sends the data as a contiguous bit stream. Byte format transfers the data as bytes, the values of which remain the same regardless of differences in byte size between the two machines (in theory - in practice you should only use this if you really know what you're doing).

CONSTRUCTOR

new ([HOST] [, OPTIONS])

This is the constructor for a new Net::FTP object. `HOST` is the name of the remote host to which an FTP connection is required.

`HOST` is optional. If `HOST` is not given then it may instead be passed as the `Host` option described below.

`OPTIONS` are passed in a hash like fashion, using key and value pairs. Possible options are:

Host - FTP host to connect to. It may be a single scalar, as defined for the `PeerAddr` option

in *IO::Socket::INET*, or a reference to an array with hosts to try in turn. The *host* method will return the value which was used to connect to the host.

Firewall - The name of a machine which acts as an FTP firewall. This can be overridden by an environment variable `FTP_FIREWALL`. If specified, and the given host cannot be directly connected to, then the connection is made to the firewall machine and the string `@hostname` is appended to the login identifier. This kind of setup is also referred to as an ftp proxy.

FirewallType - The type of firewall running on the machine indicated by **Firewall**. This can be overridden by an environment variable `FTP_FIREWALL_TYPE`. For a list of permissible types, see the description of ftp_firewall_type in *Net::Config*.

BlockSize - This is the block size that Net::FTP will use when doing transfers. (defaults to 10240)

Port - The port number to connect to on the remote machine for the FTP connection

Timeout - Set a timeout value (defaults to 120)

Debug - debug level (see the debug method in *Net::Cmd*)

Passive - If set to a non-zero value then all data transfers will be done using passive mode. If set to zero then data transfers will be done using active mode. If the machine is connected to the Internet directly, both passive and active mode should work equally well. Behind most firewall and NAT configurations passive mode has a better chance of working. However, in some rare firewall configurations, active mode actually works when passive mode doesn't. Some really old FTP servers might not implement passive transfers. If not specified, then the transfer mode is set by the environment variable `FTP_PASSIVE` or if that one is not set by the settings done by the *libnetcfg* utility. If none of these apply then passive mode is used.

Hash - If given a reference to a file handle (e.g., `\*STDERR`), print hash marks (#) on that filehandle every 1024 bytes. This simply invokes the `hash()` method for you, so that hash marks are displayed for all transfers. You can, of course, call `hash()` explicitly whenever you'd like.

LocalAddr - Local address to use for all socket connections, this argument will be passed to *IO::Socket::INET*

If the constructor fails undef will be returned and an error message will be in $@

METHODS

Unless otherwise stated all methods return either a *true* or *false* value, with *true* meaning that the operation was a success. When a method states that it returns a value, failure will be returned as *undef* or an empty list.

login ([LOGIN [,PASSWORD [, ACCOUNT]]])

Log into the remote FTP server with the given login information. If no arguments are given then the `Net::FTP` uses the `Net::Netrc` package to lookup the login information for the connected host. If no information is found then a login of *anonymous* is used. If no password is given and the login is *anonymous* then *anonymous@* will be used for password.

If the connection is via a firewall then the `authorize` method will be called with no arguments.

authorize ([AUTH [, RESP]])

This is a protocol used by some firewall ftp proxies. It is used to authorise the user to send data out. If both arguments are not specified then `authorize` uses `Net::Netrc` to do a lookup.

site (ARGS)

Send a SITE command to the remote server and wait for a response.

Returns most significant digit of the response code.

ascii

Transfer file in ASCII. CRLF translation will be done if required

binary

Transfer file in binary mode. No transformation will be done.

Hint: If both server and client machines use the same line ending for text files, then it will be faster to transfer all files in binary mode.

rename (OLDNAME, NEWNAME)

Rename a file on the remote FTP server from `OLDNAME` to `NEWNAME`. This is done by sending the RNFR and RNTO commands.

delete (FILENAME)

Send a request to the server to delete `FILENAME`.

cwd ([DIR])

Attempt to change directory to the directory given in `$dir`. If `$dir` is `".."`, the FTP `CDUP` command is used to attempt to move up one directory. If no directory is given then an attempt is made to change the directory to the root directory.

cdup ()

Change directory to the parent of the current directory.

pwd ()

Returns the full pathname of the current directory.

restart (WHERE)

Set the byte offset at which to begin the next data transfer. Net::FTP simply records this value and uses it when during the next data transfer. For this reason this method will not return an error, but setting it may cause a subsequent data transfer to fail.

rmdir (DIR [, RECURSE])

Remove the directory with the name `DIR`. If `RECURSE` is *true* then `rmdir` will attempt to delete everything inside the directory.

mkdir (DIR [, RECURSE])

Create a new directory with the name `DIR`. If `RECURSE` is *true* then `mkdir` will attempt to create all the directories in the given path.

Returns the full pathname to the new directory.

alloc (SIZE [, RECORD_SIZE])

The alloc command allows you to give the ftp server a hint about the size of the file about to be transferred using the ALLO ftp command. Some storage systems use this to make intelligent decisions about how to store the file. The `SIZE` argument represents the size of the file in bytes. The `RECORD_SIZE` argument indicates a maximum record or page size for files sent with a record or page structure.

The size of the file will be determined, and sent to the server automatically for normal files so that this method need only be called if you are transferring data from a socket, named pipe, or other stream not associated with a normal file.

ls ([DIR])

Get a directory listing of `DIR`, or the current directory.

In an array context, returns a list of lines returned from the server. In a scalar context, returns a reference to a list.

dir ([DIR])

Get a directory listing of DIR, or the current directory in long format.

In an array context, returns a list of lines returned from the server. In a scalar context, returns a reference to a list.

get (REMOTE_FILE [, LOCAL_FILE [, WHERE]])

Get REMOTE_FILE from the server and store locally. LOCAL_FILE may be a filename or a filehandle. If not specified, the file will be stored in the current directory with the same leafname as the remote file.

If WHERE is given then the first WHERE bytes of the file will not be transferred, and the remaining bytes will be appended to the local file if it already exists.

Returns LOCAL_FILE, or the generated local file name if LOCAL_FILE is not given. If an error was encountered undef is returned.

put (LOCAL_FILE [, REMOTE_FILE])

Put a file on the remote server. LOCAL_FILE may be a name or a filehandle. If LOCAL_FILE is a filehandle then REMOTE_FILE must be specified. If REMOTE_FILE is not specified then the file will be stored in the current directory with the same leafname as LOCAL_FILE.

Returns REMOTE_FILE, or the generated remote filename if REMOTE_FILE is not given.

NOTE: If for some reason the transfer does not complete and an error is returned then the contents that had been transferred will not be remove automatically.

put_unique (LOCAL_FILE [, REMOTE_FILE])

Same as put but uses the STOU command.

Returns the name of the file on the server.

append (LOCAL_FILE [, REMOTE_FILE])

Same as put but appends to the file on the remote server.

Returns REMOTE_FILE, or the generated remote filename if REMOTE_FILE is not given.

unique_name ()

Returns the name of the last file stored on the server using the STOU command.

mdtm (FILE)

Returns the *modification time* of the given file

size (FILE)

Returns the size in bytes for the given file as stored on the remote server.

NOTE: The size reported is the size of the stored file on the remote server. If the file is subsequently transferred from the server in ASCII mode and the remote server and local machine have different ideas about "End Of Line" then the size of file on the local machine after transfer may be different.

supported (CMD)

Returns TRUE if the remote server supports the given command.

hash ([FILEHANDLE_GLOB_REF],[BYTES_PER_HASH_MARK])

Called without parameters, or with the first argument false, hash marks are suppressed. If the first argument is true but not a reference to a file handle glob, then *STDERR is used. The second argument is the number of bytes per hash mark printed, and defaults to 1024. In all cases the return value is a reference to an array of two: the filehandle glob reference and the bytes per hash mark.

feature (NAME)

Determine if the server supports the specified feature. The return value is a list of lines the server responded with to describe the options that it supports for the given feature. If the feature is unsupported then the empty list is returned.

```
if ($ftp->feature( 'MDTM' )) {
  # Do something
}

if (grep { /\bTLS\b/ } $ftp->feature('AUTH')) {
  # Server supports TLS
}
```

The following methods can return different results depending on how they are called. If the user explicitly calls either of the `pasv` or `port` methods then these methods will return a *true* or *false* value. If the user does not call either of these methods then the result will be a reference to a `Net::FTP::dataconn` based object.

nlst ([DIR])

Send an `NLST` command to the server, with an optional parameter.

list ([DIR])

Same as `nlst` but using the `LIST` command

retr (FILE)

Begin the retrieval of a file called `FILE` from the remote server.

stor (FILE)

Tell the server that you wish to store a file. `FILE` is the name of the new file that should be created.

stou (FILE)

Same as `stor` but using the `STOU` command. The name of the unique file which was created on the server will be available via the `unique_name` method after the data connection has been closed.

appe (FILE)

Tell the server that we want to append some data to the end of a file called `FILE`. If this file does not exist then create it.

If for some reason you want to have complete control over the data connection, this includes generating it and calling the `response` method when required, then the user can use these methods to do so.

However calling these methods only affects the use of the methods above that can return a data connection. They have no effect on methods `get`, `put`, `put_unique` and those that do not require data connections.

port ([PORT])

Send a `PORT` command to the server. If `PORT` is specified then it is sent to the server. If not, then a listen socket is created and the correct information sent to the server.

pasv ()

Tell the server to go into passive mode. Returns the text that represents the port on which the server is listening, this text is in a suitable form to sent to another ftp server using the `port` method.

The following methods can be used to transfer files between two remote servers, providing that these two servers can connect directly to each other.

pasv_xfer (SRC_FILE, DEST_SERVER [, DEST_FILE])

This method will do a file transfer between two remote ftp servers. If `DEST_FILE` is omitted then the leaf name of `SRC_FILE` will be used.

pasv_xfer_unique (SRC_FILE, DEST_SERVER [, DEST_FILE])

Like `pasv_xfer` but the file is stored on the remote server using the STOU command.

pasv_wait (NON_PASV_SERVER)

This method can be used to wait for a transfer to complete between a passive server and a non-passive server. The method should be called on the passive server with the `Net::FTP` object for the non-passive server passed as an argument.

abort ()

Abort the current data transfer.

quit ()

Send the QUIT command to the remote FTP server and close the socket connection.

Methods for the adventurous

`Net::FTP` inherits from `Net::Cmd` so methods defined in `Net::Cmd` may be used to send commands to the remote FTP server.

quot (CMD [,ARGS])

Send a command, that Net::FTP does not directly support, to the remote server and wait for a response.

Returns most significant digit of the response code.

WARNING This call should only be used on commands that do not require data connections. Misuse of this method can hang the connection.

THE dataconn CLASS

Some of the methods defined in `Net::FTP` return an object which will be derived from this class.The dataconn class itself is derived from the `IO::Socket::INET` class, so any normal IO operations can be performed. However the following methods are defined in the dataconn class and IO should be performed using these.

read (BUFFER, SIZE [, TIMEOUT])

Read `SIZE` bytes of data from the server and place it into `BUFFER`, also performing any <CRLF> translation necessary. `TIMEOUT` is optional, if not given, the timeout value from the command connection will be used.

Returns the number of bytes read before any <CRLF> translation.

write (BUFFER, SIZE [, TIMEOUT])

Write `SIZE` bytes of data from `BUFFER` to the server, also performing any <CRLF> translation necessary. `TIMEOUT` is optional, if not given, the timeout value from the command connection will be used.

Returns the number of bytes written before any <CRLF> translation.

bytes_read ()

Returns the number of bytes read so far.

abort ()

Abort the current data transfer.

close ()

Close the data connection and get a response from the FTP server. Returns *true* if the connection was closed successfully and the first digit of the response from the server was a '2'.

UNIMPLEMENTED

The following RFC959 commands have not been implemented:

SMNT

Mount a different file system structure without changing login or accounting information.

HELP

Ask the server for "helpful information" (that's what the RFC says) on the commands it accepts.

MODE

Specifies transfer mode (stream, block or compressed) for file to be transferred.

SYST

Request remote server system identification.

STAT

Request remote server status.

STRU

Specifies file structure for file to be transferred.

REIN

Reinitialize the connection, flushing all I/O and account information.

REPORTING BUGS

When reporting bugs/problems please include as much information as possible. It may be difficult for me to reproduce the problem as almost every setup is different.

A small script which yields the problem will probably be of help. It would also be useful if this script was run with the extra options `Debug = 1>` passed to the constructor, and the output sent with the bug report. If you cannot include a small script then please include a Debug trace from a run of your program which does yield the problem.

AUTHOR

Graham Barr <gbarr@pobox.com>

SEE ALSO

Net::Netrc Net::Cmd

ftp(1), ftpd(8), RFC 959 http://www.cis.ohio-state.edu/htbin/rfc/rfc959.html

USE EXAMPLES

For an example of the use of Net::FTP see

http://www.csh.rit.edu/~adam/Progs/

`autoftp` is a program that can retrieve, send, or list files via the FTP protocol in a non-interactive manner.

CREDITS

Henry Gabryjelski <henryg@WPI.EDU> - for the suggestion of creating directories recursively.

Nathan Torkington <gnat@frii.com> - for some input on the documentation.

Roderick Schertler <roderick@gate.net> - for various inputs

COPYRIGHT