

File Transfer Protocol

File Transfer Protocol (FTP) is a [network protocol](#) used to transfer data from one computer to another through a network such as the [Internet](#).

FTP is a file transfer protocol for exchanging and manipulating files over a [TCP](#) computer network. A FTP client may connect to a FTP server to manipulate files on that server. As there are many FTP client and server programs available for different [operating systems](#), FTP is a popular choice for exchanging files independent of the operating systems involved.

The TCP/IP model (RFC 1122)
Application Layer
BGP · DHCP · DNS · FTP · Gopher · GTP · HTTP · IMAP · IRC · NNTP · NTP · POP · RIP · RPC · RTCP · RTP · RTSP · SDP · SIP · SMTP · SNMP · SOAP · SSH · SSL · STUN · Telnet · TLS · XMPP · (more)
Transport Layer
TCP · UDP · DCCP · SCTP · RSVP · ECN · (more)
Internet Layer
IP (IPv4 · IPv6) · ICMP · ICMPv6 · IGMP · IPsec · (more)
Link Layer
ARP · RARP · NDP · OSPF · Tunnels · Media Access Control · Device Drivers · (more)
<small>This box: view · talk · edit</small>

Connection methods

FTP runs exclusively over [TCP](#). It defaults to listen on [port 21](#) for incoming connections from FTP clients. A connection to this port from the FTP Client forms the control stream on which commands are passed to the FTP server from the FTP client and on occasion from the FTP server to the FTP client. FTP uses [out-of-band control](#), which means it uses a separate connection for control and data. Thus, for the actual file transfer to take place, a different connection is required which is called the data stream. Depending on the transfer mode, the process of setting up the data stream is different.

In **active mode**, the FTP client opens a [dynamic port](#), sends the FTP server the dynamic port number on which it is listening over the control stream and waits for a connection from the FTP server. When the FTP server initiates the data connection to the FTP client it binds the source port to port 20 on the FTP server.

In order to use active mode, the client sends a PORT command, with the IP and port as argument. The format for the IP and port is "h1,h2,h3,h4,p1,p2". Each

field is a decimal representation of 8 bits of the host IP, followed by the chosen data port. For example, a client with an IP of 192.168.0.1, listening on port 49154 for the data connection will send the command "PORT 192,168,0,1,192,2". The port fields should be interpreted as $p1 \times 256 + p2 = \text{port}$, or, in this example, $192 \times 256 + 2 = 49154$.

In **passive mode**, the FTP server opens a dynamic port, sends the FTP client the server's IP address to connect to and the port on which it is listening (a 16-bit value broken into a high and low byte, as explained above) over the control stream and waits for a connection from the FTP client. In this case, the FTP client binds the source port of the connection to a dynamic port.

To use passive mode, the client sends the *PASV* command to which the server would reply with something similar to "227 Entering Passive Mode (127,0,0,1,192,52)". The syntax of the IP address and port are the same as for the argument to the PORT command.

In **extended passive mode**, the FTP server operates exactly the same as passive mode, however it only transmits the port number (not broken into high and low bytes) and the client is to assume that it connects to the same IP address that was originally connected to. Extended passive mode was added by [RFC 2428](#) in September 1998.

While data is being transferred via the [data stream](#), the control stream sits idle. This can cause problems with large data transfers through [firewalls](#) which time out sessions after lengthy periods of idleness. While the file may well be successfully transferred, the control session can be disconnected by the firewall, causing an error to be generated.

The FTP protocol supports resuming of interrupted downloads using the REST command. The client passes the number of bytes it has already received as argument to the REST command and restarts the transfer. In some commandline clients for example, there is an often-ignored but valuable command, "reget" (meaning "get again") that will cause an interrupted "get" command to be continued, hopefully to completion, after a communications interruption.

Resuming uploads is not as easy. Although the FTP protocol supports the APPE command to append data to a file on the server, the client does not know the exact position at which a transfer got interrupted. It has to obtain the size of the file some other way, for example over a directory listing or using the SIZE command.

In ASCII mode (see below), resuming transfers can be troublesome if client and server use different [end of line](#) characters.

The objectives of FTP, as outlined by its [RFC](#), are:

1. To promote sharing of files (computer programs and/or data).
2. To encourage indirect or implicit use of [remote computers](#).
3. To shield a user from variations in file storage systems among different [hosts](#).
4. To transfer [data](#) reliably, and efficiently.

Criticisms of FTP

- [Passwords](#) and file contents are sent in [clear text](#), which can be intercepted by [eavesdroppers](#). There are protocol enhancements that remedy this, for instance by using SSL, TLS or [Kerberos](#).
- Multiple TCP/IP connections are used, one for the control connection, and one for each download, upload, or directory listing. Firewalls may need additional logic and/or configuration changes to account for these connections.
- It is hard to filter active mode FTP traffic on the client side by using a [firewall](#), since the client must open an arbitrary [port](#) in order to receive the connection. This problem is largely resolved by using passive mode FTP.
- It is possible to abuse the protocol's built-in proxy features to tell a [server](#) to send data to an arbitrary port of a third computer; see [FXP](#).
- FTP is a high latency protocol due to the number of commands needed to initiate a transfer.
- No integrity check on the receiver side. If a transfer is interrupted, the receiver has no way to know if the received file is complete or not. Some servers support extensions to calculate for example a file's [MD5](#) sum (e.g. using the [SITE MD5 command](#)), [XCRC](#), [XMD5](#), [XSHA](#) or [CRC checksum](#), however even then the client has to make explicit use of them. In the absence of such extensions, integrity checks have to be managed externally.
- No date/timestamp attribute transfer. Uploaded files are given a new current timestamp, unlike other file transfer protocols such as [SFTP](#), which allow attributes to be included. There is no way in the standard FTP protocol to set the time-last-modified (or time-created) datestamp that most modern filesystems preserve. There is a [draft](#) of a proposed extension that adds new commands for this, but as of yet, most of the popular FTP servers do not support it.

Security problems

The original FTP specification is an inherently insecure method of transferring files because there is no method specified for transferring data in an encrypted fashion. This means that under most network configurations, user names, passwords, FTP commands and transferred files can be "sniffed" or viewed by anyone on the same network using a [packet sniffer](#). This is a problem common to many Internet protocol specifications written prior to the creation of [SSL](#) such as

[HTTP](#), [SMTP](#) and [Telnet](#). The common solution to this problem is to use either [SFTP](#) (SSH File Transfer Protocol), or [FTPS](#) (FTP over [SSL](#)), which adds SSL or [TLS encryption](#) to FTP as specified in [RFC 4217](#).

FTP return codes

Main article: [List of FTP server return codes](#)

FTP server return codes indicate their status by the digits within them. A brief explanation of various digits' meanings are given below:

- 1xx: Positive Preliminary reply. The action requested is being initiated but there will be another reply before it begins.
- 2xx: Positive Completion reply. The action requested has been completed. The client may now issue a new command.
- 3xx: Positive Intermediate reply. The command was successful, but a further command is required before the server can act upon the request.
- 4xx: Transient Negative Completion reply. The command was not successful, but the client is free to try the command again as the failure is only temporary.
- 5xx: Permanent Negative Completion reply. The command was not successful and the client should not attempt to repeat it again.
- x0x: The failure was due to a [syntax](#) error.
- x1x: This response is a reply to a request for information.
- x2x: This response is a reply relating to connection information.
- x3x: This response is a reply relating to accounting and authorization.
- x4x: Unspecified as yet
- x5x: These responses indicate the status of the Server file system vis-a-vis the requested transfer or other file system action.

Anonymous FTP

A host which provides an FTP service may additionally provide [Anonymous](#) FTP access as well. Under this arrangement, users do not strictly need an [account](#) on the host. Instead the user typically enters 'anonymous' or 'ftp' when prompted for username. Although users are commonly asked to send their [email](#) address as their password, little to no verification is actually performed on the supplied data.

As modern FTP clients typically hide the anonymous login process from the user, the ftp client will supply dummy data as the password (since the user's email address may not be known to the application). For example, the following ftp [user agents](#) specify the listed passwords for anonymous logins:

- [Mozilla Firefox](#) (2.0) — mozilla@example.com
- [KDE Konqueror](#) (3.5) — anonymous@

- [wget](#) (1.10.2) — wget@
- [lftp](#) (3.4.4) — lftp@

The [Gopher protocol](#) has been suggested as an alternative to anonymous FTP, as well as [Trivial File Transfer Protocol](#) and [File Service Protocol](#).^[*citation needed*]

Data format

While transferring data over the network, several data representations can be used. The two most common transfer modes are:

1. [ASCII](#) mode
2. [Binary](#) mode: In "Binary mode", the sending machine sends each file [byte](#) for byte and as such the recipient stores the bytestream as it receives it. (The FTP standard calls this "IMAGE" or "I" mode)

In "ASCII mode", any form of data that is not plain text will be corrupted. When a file is sent using an ASCII-type transfer, the individual letters, numbers, and characters are sent using their ASCII character codes. The receiving machine saves these in a text file in the appropriate format (for example, a Unix machine saves it in a Unix format, a Windows machine saves it in a Windows format). Hence if an ASCII transfer is used it can be assumed [plain text](#) is sent, which is stored by the receiving computer in its own format. Translating between text formats might entail substituting the [end of line](#) and [end of file](#) characters used on the source platform with those on the destination platform, e.g. a Windows machine receiving a file from a Unix machine will replace the [line feeds](#) with [carriage return](#)-line feed pairs. It might also involve translating characters; for example, when transferring from an [IBM mainframe](#) to a system using ASCII, [EBCDIC](#) characters used on the mainframe will be translated to their ASCII equivalents, and when transferring from the system using ASCII to the mainframe, ASCII characters will be translated to their EBCDIC equivalents.

By default, most FTP clients use ASCII mode. Some clients try to determine the required transfer-mode by inspecting the file's name or contents, or by determining whether the server is running an operating system with the same text file format.

The FTP specifications also list the following transfer modes:

1. EBCDIC mode - this transfers bytes, except they are encoded in EBCDIC rather than ASCII. Thus, for example, the ASCII mode server
2. Local mode - this is designed for use with systems that are word-oriented rather than byte-oriented. For example mode "L 36" can be used to transfer binary data between two [36-bit](#) machines. In L mode, the words are packed into bytes rather than being padded. Given the predominance

of byte-oriented hardware nowadays, this mode is rarely used. However, some FTP servers accept "L 8" as being equivalent to "I".

In practice, these additional transfer modes are rarely used. They are however still used by some [legacy mainframe](#) systems.

The text (ASCII/EBCDIC) modes can also be qualified with the type of carriage control used (e.g. TELNET NVT carriage control, ASA carriage control), although that is rarely used nowadays.

Note that the terminology "mode" is technically incorrect, although commonly used by FTP clients. "MODE" in [RFC 959](#) refers to the format of the protocol data stream (STREAM, BLOCK or COMPRESSED), as opposed to the format of the underlying file. What is commonly called "mode" is actually the "TYPE", which specifies the format of the file rather than the data stream. FTP also supports specification of the file structure ("STRU"), which can be either FILE (stream-oriented files), RECORD (record-oriented files) or PAGE (special type designed for use with TENEX). PAGE STRU is not really useful for non-TENEX systems, and RFC1123 section 4.1.2.3 recommends that it not be implemented.

FTP and web browsers

Most recent [web browsers](#) and [file managers](#) can connect to FTP servers, although they may lack the support for protocol extensions such as [FTPS](#). This allows manipulation of remote files over FTP through an interface similar to that used for local files. This is done via an FTP [URL](#), which takes the form `ftp(s)://<ftpserveraddress>` (e.g., <ftp://ftp.gimp.org/>). A password can optionally be given in the URL, e.g.:

`ftp(s)://<login>:<password>@<ftpserveraddress>:<port>`. Most web-browsers require the use of passive mode FTP, which not all FTP servers are capable of handling. Some browsers allow only the downloading of files, but offer no way to upload files to the server.

FTP and NAT devices

The representation of the IPs and ports in the PORT command and PASV reply poses another challenge for [NAT](#) devices in handling FTP. The NAT device must alter these values, so that they contain the IP of the NAT-ed client, and a port chosen by the NAT device for the data connection. The new IP and port will probably differ in length in their decimal representation from the original IP and port. This means that altering the values on the control connection by the NAT device must be done carefully, changing the [TCP](#) Sequence and Acknowledgment fields for all subsequent packets.

For example: A client with an IP of 192.168.0.1, starting an active mode transfer on port 1025, will send the string "PORT 192,168,0,1,4,1". A NAT device masquerading this client with an IP of 192.168.15.5, with a chosen port of 2000 for the data connection, will need to replace the above string with "PORT 192,168,15,5,7,208".

The new string is 23 characters long, compared to 20 characters in the original packet. The Acknowledgment field by the server to this packet will need to be decreased by 3 bytes by the NAT device for the client to correctly understand that the PORT command has arrived to the server. If the NAT device is not capable of correcting the Sequence and Acknowledgement fields, it will not be possible to use active mode FTP. Passive mode FTP will work in this case, because the information about the IP and port for the data connection is sent by the server, which doesn't need to be NATed. If NAT is performed on the server by the NAT device, then the exact opposite will happen. Active mode will work, but passive mode will fail.

It should be noted that many NAT devices perform this protocol inspection and modify the PORT command without being explicitly told to do so by the user. This can lead to several problems. First of all, there is no guarantee that the used protocol really is FTP, or it might use some extension not understood by the NAT device. One example would be an SSL secured FTP connection. Due to the encryption, the NAT device will be unable to modify the address. As result, active mode transfers will fail only if encryption is used, much to the confusion of the user.

The proper way to solve this is to tell the client which IP address and ports to use for active mode. Furthermore, the NAT device has to be configured to forward the selected range of ports to the client's machine.

See also [Application-level gateway](#)

FTP over SSH (SFTP)

FTP over SSH (SFTP) refers to the practice of tunneling a normal FTP session over an [SSH](#) connection.

Because FTP uses multiple [TCP](#) connections (unusual for a TCP/IP protocol that is still in use), it is particularly difficult to tunnel over SSH. With many SSH clients, attempting to set up a tunnel for the *control channel* (the initial client-to-server connection on port 21) will protect only that channel; when data is transferred, the FTP software at either end will set up new TCP connections (*data channels*) which will bypass the SSH connection, and thus have no [confidentiality](#), [integrity protection](#), etc.

If the FTP client is configured to use *passive mode* and to connect to a [SOCKS](#) server interface that many SSH clients can present for tunneling, it is possible to run all the FTP channels over the SSH connection.

Otherwise, it is necessary for the SSH client software to have specific knowledge of the FTP protocol, and monitor and rewrite FTP control channel messages and autonomously open new forwardings for FTP data channels. Version 3 of [SSH Communications Security](#)'s software suite, and the [GPL](#) licensed [FONC](#) are two software packages that support this mode.

FTP over SSH is sometimes referred to as **secure FTP**; this should not be confused with other methods of securing FTP, such as with SSL/TLS ([FTPS](#)). Other methods of transferring files using SSH that are not related to FTP include [SFTP](#) and [SCP](#); in each of these, the entire conversation (credentials and data) is always protected by the SSH protocol.

See also

- [FTAM](#)
- [FTPFS](#)
- [List of FTP server return codes](#)
- [List of FTP commands](#)
- [List of file transfer protocols](#)
- [OBEX](#)
- [Shared file access](#)
- [TCP Wrapper](#)
- [Comparison of FTP client software](#)
- [List of FTP server software](#)
- [Comparison of FTP server software](#)

Further reading

The protocol is [standardized](#) in [RFC](#) 959 by the [IETF](#) as:

- [RFC 959](#) File Transfer Protocol (FTP). J. Postel, J. Reynolds. Oct-1985. This obsoleted the preceding [RFC 765](#) and earlier FTP RFCs back to the original [RFC 114](#).
- [RFC 1579](#) Firewall-Friendly FTP.
- [RFC 2228](#) — FTP Security Extensions
- [RFC 2428](#) — Extensions for IPv6, NAT, and Extended passive mode Sep-1998.
- [RFC 3659](#) — Extensions to FTP. P. Hethmon. March-2007.

External links

- [FTP Reviewed](#) — a review of the protocol notably from a security standpoint
- [Raw FTP command list](#)
- [FTP Sequence Diagram](#) (in [PDF](#) format)

Retrieved from "http://en.wikipedia.org/wiki/File_Transfer_Protocol"