NAME

zshzftpsys − zftp function front−end

DESCRIPTION

This describes the set of shell functions supplied with the source distribution as an interface to the **zftp** builtin command, allowing you to perform FTP operations from the shell command line or within functions or scripts. The interface is similar to a traditional FTP client (e.g. the **ftp** command itself, see *ftp*(1)), but as it is entirely done within the shell all the familiar completion, editing and globbing features, and so on, are present, and macros are particularly simple to write as they are just ordinary shell functions.

The prerequisite is that the **zftp** command, as described in *zshmodules*(1) , must be available in the version of **zsh** installed at your site. If the shell is configured to load new commands at run time, it probably is: typing '**zmodload zsh/zftp**' will make sure (if that runs silently, it has worked). If this is not the case, it is possible **zftp** was linked into the shell anyway: to test this, type '**which zftp**' and if **zftp** is available you will get the message '**zftp: shell built−in command**'.

Commands given directly with **zftp** builtin may be interspersed between the functions in this suite; in a few cases, using **zftp** directly may cause some of the status information stored in shell parameters to become invalid. Note in particular the description of the variables **$ZFTP_TMOUT**, **$ZFTP_PREFS** and **$ZFTP_VERBOSE** for **zftp**.

INSTALLATION

You should make sure all the functions from the **Functions/Zftp** directory of the source distribution are available; they all begin with the two letters '**zf**'. They may already have been installed on your system; otherwise, you will need to find them and copy them. The directory should appear as one of the elements of the **$fpath** array (this should already be the case if they were installed), and at least the function **zfinit** should be autoloaded; it will autoload the rest. Finally, to initialize the use of the system you need to call the **zfinit** function. The following code in your **.zshrc** will arrange for this; assume the functions are stored in the directory **~/myfns**:

```
fpath=(~/myfns $fpath)
autoload −U zfinit
zfinit
```

Note that **zfinit** assumes you are using the **zmodload** method to load the **zftp** command. If it is already built into the shell, change **zfinit** to **zfinit −n**. It is helpful (though not essential) if the call to **zfinit** appears after any code to initialize the new completion system, else unnecessary **compctl** commands will be given.

FUNCTIONS

The sequence of operations in performing a file transfer is essentially the same as that in a standard FTP client. Note that, due to a quirk of the shell's **getopts** builtin, for those functions that handle options you must use '**−−**' rather than '**−**' to ensure the remaining arguments are treated literally (a single '**−**' is treated as an argument).

Opening a connection

zfparams [*host* [*user* [*password ...*]]]

Set or show the parameters for a future **zfopen** with no arguments. If no arguments are given, the current parameters are displayed (the password will be shown as a line of asterisks). If a *host* is given, and either the *user* or *password* is not, they will be prompted for; also, any parameter given as '**?**' will be prompted for, and if the '**?**' is followed by a string, that will be used as the prompt. As **zfopen** calls **zfparams** to store the parameters, this usually need not be called directly.

A single argument '**−**' will delete the stored parameters. This will also cause the memory of the last directory (and so on) on the other host to be deleted.

zfopen [**−1**] [*host* [*user* [*password* [*account*]]]]

If *host* is present, open a connection to that host under username *user* with password *password* (and, on the rare occasions when it is necessary, account *account*). If a necessary parameter is missing or given as '**?**' it will be prompted for. If *host* is not present, use a previously stored set of parameters.

If the command was successful, and the terminal is compatible with **xterm** or is **sun−cmd**, a summary will appear in the title bar, giving the local **host:directory** and the remote **host:directory**; this is handled by the function **zftp_chpwd**, described below.

Normally, the *host*, *user* and *password* are internally recorded for later re−opening, either by a **zfopen** with no arguments, or automatically (see below). With the option '**−1**', no information is stored. Also, if an open command with arguments failed, the parameters will not be retained (and any previous parameters will also be deleted). A **zfopen** on its own, or a **zfopen −1**, never alters the stored parameters.

Both **zfopen** and **zfanon** (but not **zfparams**) understand URLs of the form **ftp://***host*/*path*... as meaning to connect to the *host*, then change directory to *path* (which must be a directory, not a file). The '**ftp://**' can be omitted; the trailing '**/**' is enough to trigger recognition of the *path*. Note prefixes other than '**ftp:**' are not recognized, and that all characters after the first slash beyond *host* are significant in *path*.

zfanon [**−1**] *host*
Open a connection *host* for anonymous FTP. The username used is '**anonymous**'. The password (which will be reported the first time) is generated as *user*@*host*; this is then stored in the shell parameter **$EMAIL_ADDR** which can alternatively be set manually to a suitable string.

Directory management

zfcd [*dir*]
zfcd −
zfcd *old new*
Change the current directory on the remote server: this is implemented to have many of the features of the shell builtin **cd**.

In the first form with *dir* present, change to the directory *dir*. The command '**zfcd ..**' is treated specially, so is guaranteed to work on non−UNIX servers (note this is handled internally by **zftp**). If *dir* is omitted, has the effect of '**zfcd ~**'.

The second form changes to the directory previously current.

The third form attempts to change the current directory by replacing the first occurrence of the string *old* with the string *new* in the current directory.

Note that in this command, and indeed anywhere a remote filename is expected, the string which on the local host corresponds to '~' is converted back to a '~' before being passed to the remote machine. This is convenient because of the way expansion is performed on the command line before **zfcd** receives a string. For example, suppose the command is '**zfcd ~/foo**'. The shell will expand this to a full path such as '**zfcd /home/user2/pws/foo**'. At this stage, **zfcd** recognises the initial path as corresponding to '~' and will send the directory to the remote host as **~/foo**, so that the '~' will be expanded by the server to the correct remote host directory. Other named directories of the form '**~name**' are not treated in this fashion.

zfhere Change directory on the remote server to the one corresponding to the current local directory, with special handling of '~' as in **zfcd**. For example, if the current local directory is **~/foo/bar**, then **zfhere** performs the effect of '**zfcd ~/foo/bar**'.

zfdir [**−rfd**] [**−**] [*dir−options*] [*dir*]
Produce a long directory listing. The arguments *dir−options* and *dir* are passed directly to the server and their effect is implementation dependent, but specifying a particular remote directory *dir* is usually possible. The output is passed through a pager given by the environment variable **$PAGER**, or '**more**' if that is not set.

The directory is usually cached for re−use. In fact, two caches are maintained. One is for use when there is no *dir−options* or *dir*, i.e. a full listing of the current remote directory; it is flushed when the current remote directory changes. The other is kept for repeated use of **zfdir** with the same arguments; for example, repeated use of '**zfdir /pub/gnu**' will only require the directory to be retrieved on the first call. Alternatively, this cache can be re−viewed with the **−r** option. As

relative directories will confuse **zfdir**, the **−f** option can be used to force the cache to be flushed before the directory is listed. The option **−d** will delete both caches without showing a directory listing; it will also delete the cache of file names in the current remote directory, if any.

zfls [*ls−options*] [*dir*]
List files on the remote server. With no arguments, this will produce a simple list of file names for the current remote directory. Any arguments are passed directly to the server. No pager and no caching is used.

Status commands

zftype [*type*]
With no arguments, show the type of data to be transferred, usually ASCII or binary. With an argument, change the type: the types '**A**' or '**ASCII**' for ASCII data and '**B**' or '**BINARY**', '**I**' or '**IMAGE**' for binary data are understood case−insensitively.

zfstat [**−v**]
Show the status of the current or last connection, as well as the status of some of **zftp**'s status variables. With the **−v** option, a more verbose listing is produced by querying the server for its version of events, too.

Retrieving files

The commands for retrieving files all take at least two options. **−G** suppresses remote filename expansion which would otherwise be performed (see below for a more detailed description of that). **−t** attempts to set the modification time of the local file to that of the remote file: see the description of the function **zfrtime** below for more information.

zfget [**−Gtc**] *file1 ...*
Retrieve all the listed files *file1 ...* one at a time from the remote server. If a file contains a '**/**', the full name is passed to the remote server, but the file is stored locally under the name given by the part after the final '**/**'. The option **−c** (cat) forces all files to be sent as a single stream to standard output; in this case the **−t** option has no effect.

zfuget [**−Gvst**] *file1 ...*
As **zfget**, but only retrieve files where the version on the remote server is newer (has a later modification time), or where the local file does not exist. If the remote file is older but the files have different sizes, or if the sizes are the same but the remote file is newer, the user will usually be queried. With the option **−s**, the command runs silently and will always retrieve the file in either of those two cases. With the option **−v**, the command prints more information about the files while it is working out whether or not to transfer them.

zfcget [**−Gt**] *file1 ...*
As **zfget**, but if any of the local files exists, and is shorter than the corresponding remote file, the command assumes that it is the result of a partially completed transfer and attempts to transfer the rest of the file. This is useful on a poor connection which keeps failing.

Note that this requires a commonly implemented, but non−standard, version of the FTP protocol, so is not guaranteed to work on all servers.

zfgcp [**−Gt**] *remote−file local−file*
zfgcp [**−Gt**] *rfile1 ... ldir*
This retrieves files from the remote server with arguments behaving similarly to the **cp** command.

In the first form, copy *remote−file* from the server to the local file *local−file*.

In the second form, copy all the remote files *rfile1 ...* into the local directory *ldir* retaining the same basenames. This assumes UNIX directory semantics.

Sending files

zfput [**−r**] *file1 ...*
Send all the *file1 ...* given separately to the remote server. If a filename contains a '**/**', the full filename is used locally to find the file, but only the basename is used for the remote file name.

With the option **−r**, if any of the *files* are directories they are sent recursively with all their subdirectories, including files beginning with '**.**'. This requires that the remote machine understand UNIX file semantics, since '**/**' is used as a directory separator.

zfuput [**−vs**] *file1 ...*
As **zfput**, but only send files which are newer than their remote equivalents, or if the remote file does not exist. The logic is the same as for **zfuget**, but reversed between local and remote files.

zfcput *file1 ...*
As **zfput**, but if any remote file already exists and is shorter than the local equivalent, assume it is the result of an incomplete transfer and send the rest of the file to append to the existing part. As the FTP append command is part of the standard set, this is in principle more likely to work than **zfcget**.

zfpcp *local−file remote−file*
zfpcp *lfile1 ... rdir*
This sends files to the remote server with arguments behaving similarly to the **cp** command.

With two arguments, copy *local−file* to the server as *remote−file*.

With more than two arguments, copy all the local files *lfile1 ...* into the existing remote directory *rdir* retaining the same basenames. This assumes UNIX directory semantics.

A problem arises if you attempt to use **zfpcp** *lfile1 rdir*, i.e. the second form of copying but with two arguments, as the command has no simple way of knowing if *rdir* corresponds to a directory or a filename. It attempts to resolve this in various ways. First, if the *rdir* argument is '**.**' or '**..**' or ends in a slash, it is assumed to be a directory. Secondly, if the operation of copying to a remote file in the first form failed, and the remote server sends back the expected failure code 553 and a reply including the string '**Is a directory**', then **zfpcp** will retry using the second form.

Closing the connection

zfclose Close the connection.

Session management

zfsession [**−lvod**] [*sessname*]
Allows you to manage multiple FTP sessions at once. By default, connections take place in a session called '**default**'; by giving the command '**zfsession** *sessname*' you can change to a new or existing session with a name of your choice. The new session remembers its own connection, as well as associated shell parameters, and also the host/user parameters set by **zfparams**. Hence you can have different sessions set up to connect to different hosts, each remembering the appropriate host, user and password.

With no arguments, **zfsession** prints the name of the current session; with the option **−l** it lists all sessions which currently exist, and with the option **−v** it gives a verbose list showing the host and directory for each session, where the current session is marked with an asterisk. With **−o**, it will switch to the most recent previous session.

With **−d**, the given session (or else the current one) is removed; everything to do with it is completely forgotten. If it was the only session, a new session called '**default**' is created and made current. It is safest not to delete sessions while background commands using **zftp** are active.

zftransfer *sess1***:***file1 sess2***:***file2*
Transfer files between two sessions; no local copy is made. The file is read from the session *sess1* as *file1* and written to session *sess2* as file *file2*; *file1* and *file2* may be relative to the current directories of the session. Either *sess1* or *sess2* may be omitted (though the colon should be retained if there is a possibility of a colon appearing in the file name) and defaults to the current session; *file2* may be omitted or may end with a slash, in which case the basename of *file1* will be added. The sessions *sess1* and *sess2* must be distinct.

The operation is performed using pipes, so it is required that the connections still be valid in a subshell, which is not the case under versions of some operating systems, presumably due to a system bug.

Bookmarks

The two functions **zfmark** and **zfgoto** allow you to 'bookmark' the present location (host, user and directory) of the current FTP connection for later use. The file to be used for storing and retrieving bookmarks is given by the parameter **$ZFTP_BMFILE**; if not set when one of the two functions is called, it will be set to the file **.zfbkmarks** in the directory where your zsh startup files live (usually ~).

zfmark [*bookmark*]

If given an argument, mark the current host, user and directory under the name *bookmark* for later use by **zfgoto**. If there is no connection open, use the values for the last connection immediately before it was closed; it is an error if there was none. Any existing bookmark under the same name will be silently replaced.

If not given an argument, list the existing bookmarks and the points to which they refer in the form *user*@*host*:*directory*; this is the format in which they are stored, and the file may be edited directly.

zfgoto [**−n**] *bookmark*

Return to the location given by *bookmark*, as previously set by **zfmark**. If the location has user '**ftp**' or '**anonymous**', open the connection with **zfanon**, so that no password is required. If the user and host parameters match those stored for the current session, if any, those will be used, and again no password is required. Otherwise a password will be prompted for.

With the option **−n**, the bookmark is taken to be a nickname stored by the **ncftp** program in its bookmark file, which is assumed to be **~/.ncftp/bookmarks**. The function works identically in other ways. Note that there is no mechanism for adding or modifying **ncftp** bookmarks from the zftp functions.

Other functions

Mostly, these functions will not be called directly (apart from **zfinit**), but are described here for completeness. You may wish to alter **zftp_chpwd** and **zftp_progress**, in particular.

zfinit [**−n**]

As described above, this is used to initialize the zftp function system. The **−n** option should be used if the zftp command is already built into the shell.

zfautocheck [**−dn**]

This function is called to implement automatic reopening behaviour, as described in more detail below. The options must appear in the first argument; **−n** prevents the command from changing to the old directory, while **−d** prevents it from setting the variable **do_close**, which it otherwise does as a flag for automatically closing the connection after a transfer. The host and directory for the last session are stored in the variable **$zflastsession**, but the internal host/user/password parameters must also be correctly set.

zfcd_match *prefix suffix*

This performs matching for completion of remote directory names. If the remote server is UNIX, it will attempt to persuade the server to list the remote directory with subdirectories marked, which usually works but is not guaranteed. On other hosts it simply calls **zfget_match** and hence completes all files, not just directories. On some systems, directories may not even look like filenames.

zfget_match *prefix suffix*

This performs matching for completion of remote filenames. It caches files for the current directory (only) in the shell parameter **$zftp_fcache**. It is in the form to be called by the **−K** option of **compctl**, but also works when called from a widget−style completion function with *prefix* and *suffix* set appropriately.

zfrglob *varname*

Perform remote globbing, as describes in more detail below. *varname* is the name of a variable containing the pattern to be expanded; if there were any matches, the same variable will be set to the expanded set of filenames on return.

zfrtime *lfile rfile* [*time*]
: Set the local file *lfile* to have the same modification time as the remote file *rfile*, or the explicit time *time* in FTP format **CCYYMMDDhhmmSS** for the GMT timezone. This uses the shell's **zsh/datetime** module to perform the conversion from GMT to local time.

zftp_chpwd
: This function is called every time a connection is opened, or closed, or the remote directory changes. This version alters the title bar of an **xterm**−compatible or **sun−cmd** terminal emulator to reflect the local and remote hostnames and current directories. It works best when combined with the function **chpwd**. In particular, a function of the form

```
chpwd() {
  if [[ −n $ZFTP_USER ]]; then
    zftp_chpwd
  else
    # usual chpwd e.g put host:directory in title bar
  fi
}
```

  fits in well.

zftp_progress
: This function shows the status of the transfer. It will not write anything unless the output is going to a terminal; however, if you transfer files in the background, you should turn off progress reports by hand using '**zstyle ':zftp:*' progress none**'. Note also that if you alter it, any output *must* be to standard error, as standard output may be a file being received. The form of the progress meter, or whether it is used at all, can be configured without altering the function, as described in the next section.

zffcache
: This is used to implement caching of files in the current directory for each session separately. It is used by **zfget_match** and **zfrglob**.

MISCELLANEOUS FEATURES

Configuration

Various styles are available using the standard shell style mechanism, described in *zshmodules*(1). Briefly, the command '**zstyle ':zftp:*' *style value ...*'. defines the *style* to have value *value*; more than one value may be given, although that is not useful in the cases described here. These values will then be used throughout the zftp function system. For more precise control, the first argument, which gives a context in which the style applies, can be modified to include a particular function, as for example '**:zftp:zfget**': the style will then have the given value only in the **zfget** function. Values for the same style in different contexts may be set; the most specific function will be used, where strings are held to be more specific than patterns, and longer patterns and shorter patterns. Note that only the top level function name, as called by the user, is used; calling of lower level functions is transparent to the user. Hence modifications to the title bar in **zftp_chpwd** use the contexts **:zftp:zfopen**, **:zftp:zfcd**, etc., depending where it was called from. The following styles are understood:

progress
: Controls the way that **zftp_progress** reports on the progress of a transfer. If empty, unset, or '**none**', no progress report is made; if '**bar**' a growing bar of inverse video is shown; if '**percent**' (or any other string, though this may change in future), the percentage of the file transferred is shown. The bar meter requires that the width of the terminal be available via the **$COLUMNS** parameter (normally this is set automatically). If the size of the file being transferred is not available, **bar** and **percent** meters will simply show the number of bytes transferred so far.

  When **zfinit** is run, if this style is not defined for the context **:zftp:***, it will be set to 'bar'.

update Specifies the minimum time interval between updates of the progress meter in seconds. No update is made unless new data has been received, so the actual time interval is limited only by **$ZFTP_TIMEOUT**.

As described for **progress**, **zfinit** will force this to default to 1.

remote−glob
If set to '**1**', '**yes**' or '**true**', filename generation (globbing) is performed on the remote machine instead of by zsh itself; see below.

titlebar
If set to '**1**', '**yes**' or '**true**', **zftp_chpwd** will put the remote host and remote directory into the titlebar of terminal emulators such as xterm or sun−cmd that allow this.

As described for **progress**, **zfinit** will force this to default to 1.

chpwd If set to '**1**' '**yes**' or '**true**', **zftp_chpwd** will call the function **chpwd** when a connection is closed. This is useful if the remote host details were put into the terminal title bar by **zftp_chpwd** and your usual **chpwd** also modifies the title bar.

When **zfinit** is run, it will determine whether **chpwd** exists and if so it will set the default value for the style to 1 if none exists already.

Note that there is also an associative array **zfconfig** which contains values used by the function system. This should not be modified or overwritten.

Remote globbing

The commands for retrieving files usually perform filename generation (globbing) on their arguments; this can be turned off by passing the option **−G** to each of the commands. Normally this operates by retrieving a complete list of files for the directory in question, then matching these locally against the pattern supplied. This has the advantage that the full range of zsh patterns (respecting the setting of the option **EXTENDED_GLOB**) can be used. However, it means that the directory part of a filename will not be expanded and must be given exactly. If the remote server does not support the UNIX directory semantics, directory handling is problematic and it is recommended that globbing only be used within the current directory. The list of files in the current directory, if retrieved, will be cached, so that subsequent globs in the same directory without an intervening **zfcd** are much faster.

If the **remote−glob** style (see above) is set, globbing is instead performed on the remote host: the server is asked for a list of matching files. This is highly dependent on how the server is implemented, though typically UNIX servers will provide support for basic glob patterns. This may in some cases be faster, as it avoids retrieving the entire list of directory contents.

Automatic and temporary reopening

As described for the **zfopen** command, a subsequent **zfopen** with no parameters will reopen the connection to the last host (this includes connections made with the **zfanon** command). Opened in this fashion, the connection starts in the default remote directory and will remain open until explicitly closed.

Automatic re−opening is also available. If a connection is not currently open and a command requiring a connection is given, the last connection is implicitly reopened. In this case the directory which was current when the connection was closed again becomes the current directory (unless, of course, the command given changes it). Automatic reopening will also take place if the connection was close by the remote server for whatever reason (e.g. a timeout). It is not available if the **−1** option to **zfopen** or **zfanon** was used.

Furthermore, if the command issued is a file transfer, the connection will be closed after the transfer is finished, hence providing a one−shot mode for transfers. This does not apply to directory changing or listing commands; for example a **zfdir** may reopen a connection but will leave it open. Also, automatic closure will only ever happen in the same command as automatic opening, i.e a **zfdir** directly followed by a **zfget** will never close the connection automatically.

Information about the previous connection is given by the **zfstat** function. So, for example, if that reports:

Session: default
Not connected.
Last session: ftp.bar.com:/pub/textfiles

then the command **zfget file.txt** will attempt to reopen a connection to **ftp.bar.com**, retrieve the file **/pub/textfiles/file.txt**, and immediately close the connection again. On the other hand, **zfcd ..** will open the

connection in the directory **/pub** and leave it open.

Note that all the above is local to each session; if you return to a previous session, the connection for that session is the one which will be reopened.

Completion

Completion of local and remote files, directories, sessions and bookmarks is supported. The older, **compctl**−style completion is defined when **zfinit** is called; support for the new widget−based completion system is provided in the function **Completion/Zsh/Command/_zftp**, which should be installed with the other functions of the completion system and hence should automatically be available.