UFTP STANDALONE CLIENT

UNICORE Team

Document Version:	1.0.0
Component Version:	1.3.2
Date:	18 09 2020

Contents

This is a commandline client for UFTP. It allows to

- list remote directories
- upload/download files
- sync files
- make remote directories
- delete remote files or directories
- manage shares and access shared data

The *uftp* client will connect to an authentication server (either a UNICORE/X server or the UFTP Auth server) to authenticate and then to the *uftpd* server for transferring data or making a file operation.

The *uftp* client supports username/password authentication, OIDC token authentication and (on UNIX) ssh-key authentication.

The *uftp* client supports multiple concurrent FTP connections for highly efficient data transfers in high-performance environments and multiple TCP streams per connection for networks with high delay.

1 Prerequisites

- Java 8 or later (OpenJDK preferred)
- Access to a UFTP authentication service and the corresponding UFTPD server. To use the client, you need to know the address of the authentication service.
- You need valid credentials for the UFTP authentication service

2 Installation

If you use the rpm or deb package, install it using the package manager of your Linux distribution.

If using the zip or tar archive, unpack it in a location of your choice. Add the *bin* directory to your path. (Alternatively, you can link or copy the *bin/uftp* script to a directory that is already on your path, in this case edit the script and setup the required directories.)

3 Basic Usage

In this manual, we use the following format to indicate commands that you can type on the command line:

```
$> some_command
```

and assume that the bin directory of the UFTP client is on your path.

Invoking uftp without any arguments,

```
$> uftp
```

will list the available commands.

Invoking

```
$> uftp <command> -h
```

will show help for a particular command

Invoking

```
$> uftp -version
```

will show version information.

4 Authentication

By default, the *uftp* client will use the current username ($USER) with SSH key authentication to authenticate to the Auth server.

You can set a different default username via the UFTP_USER environment variable. This is useful if your remote username is not the same as your local username.

The *uftp* client supports various means of authentication. Depending on the server you want to access, you can choose from

- user name with SSH key (default)
- user name with password
- oidc-agent (see https://github.com/indigo-dc/oidc-agent)
- manual specification of the HTTP Authorization header value

To explicitely specify the remote username, use the "-u <username>" option, e.g.

```
$> uftp ls -u username https://localhost:9000/rest/auth/TEST:/home/ ↩
    demo/
```

The credentials can be given in multiple ways.

- On the command line "-u username:password"

```
$> uftp ls -u username:password ...
```

- You can tell the uftp client to query the password interactively by giving the "-P" option, e.g.

```
$> uftp ls -u username -P ...
```

- If no password is given, the client will attempt to use an SSH key for authentication, this has to be configured on the authentication server accordingly. If you have multiple keys, use the "-i" option to select one. Otherwise, the client will check "/.uftp/" and "/.ssh/" for useable keys. The SSH agent is supported, too.
- The very useful *oidc-agent* tool is also directly supported via *-O <account_name>*. In this case no username is required.

```
$> uftp ls -O hbp ...
```

- Last not least you can directly specify a value for the HTTP *Authorization* header with the "-A" option. This allows to use an OIDC bearer token for authorization, e.g. `-A "Bearer <oidc_token>"`. In this case no username is required.

```
$> uftp ls -A "Bearer <oidc_token>" ...
```

5 Usage

In the following usage examples, the authentication service is located at "localhost:9000/rest/auth/" and the user name is *username*. Replace these values by the correct ones for your installation.

5.1 Listing a directory: the "ls" command

```
$> uftp ls https://localhost:9000/rest/auth/TEST:/home/demo/
```

will list the */home/demo* directory.

5.2 Copying data: the "cp" command

The *cp* command is used to copy local data to a remote server or vice versa. Remote locations are indicated by the "https://" prefix, and you need your user name, and the URL of the authentication server.

It has a number of features, which will be shown in the following.

5.2.1 Basic usage

Downloading a single file:

```
$> uftp cp https://localhost:9000/rest/auth/TEST:/home/demo/test. ↩
    data .
```

will download the */home/demo/test.data* file to the current directory

Download files using wildcards:

```
 $> uftp cp https://localhost:9000/rest/auth/TEST:/home/demo/data/* ↩
      .
```

will download all files in the */home/demo/test* directory to the current directory

Similar commands work for upload.

Uploading files using wildcards:

```
 $> uftp cp "/data/*" https://localhost:9000/rest/auth/TEST:/home/ ↩
     demo/data/ .
```

Note that wildcards should be escaped to avoid the shell doing the expansion, which will also work, but generally be slower.

The recurse flag, *-r*, tells uftp to also copy subdirectories.

5.2.2 Piping data

The "cp" command can read/write from the console streams, which is great for integrating uftp into Unix pipes. The "-" is used as a special "file name" to indicate that data should be read/written using the console.

5.2.3 Transferring with tar and zip

For example to tar the contents of a directory and upload the tar file using uftp:

```
$> tar cz dir/* | uftp cp - https://localhost:9000/rest/auth/TEST:/ ↩
    archive.tgz
```

With recent servers (UFTPD 2.7.0 and later), the UFTPD server can also unpack tar and zip streams, this is very useful to efficiently transfer many small files. To enable this, add the *-a* option, and DO NOT compress the tar stream.

```
$> tar c dir/* | uftp cp -a - https://localhost:9000/rest/auth/TEST ↩
    :/target_location/
```

or, using zip

```
$> zip -r - dir/* | uftp cp -a - https://localhost:9000/rest/auth/ ↩
    TEST:/target_location/
```

Note that zip will compress data, so might be slower or faster than tar, depending on network bandwith and processing speed.

Similarly, "-" can be used to write data to standard output. As an example, consider this:

```
$> uftp cp https://localhost:9000/rest/auth/TEST:/archive.tgz - |  ↩
    tar tz
```

Or use uftp to cat a remote file:

```
$> uftp cp https://localhost:9000/rest/auth/TEST:/foo.txt -
```

5.2.4 Using multiple FTP connections

When transferring large files (or many files) over a high-performance network, performance can be vastly improved by using multiple FTP connections. (NOTE this is different from the multiple TCP streams as set via the *-n* option).

Use the "-t" option to set the desired number of streams. Note that the server may have a limit on the allowed number of concurrent connections, if in doubt, ask your server administrator.

```
$> uftp cp -t 2 https://localhost:9000/rest/auth/TEST:/home/demo/*  ↩
    .
```

Files larger than a certain size will be transferred concurrently using more than one stream. This threshold size is 512MB, but you can set it to a different value using the *-T* option. For example, to split files larger than 1MB

```
$> uftp cp -t 2 -T 1M https://localhost:9000/rest/auth/TEST:/home/ ↩
    demo/* .
```

5.2.5 Byte ranges

To copy just part of a file, a byte range can be given with the "-R" option. Counting starts at "zero". For example to download only the first 1024 bytes of file (i.e. the range 0 - 1023), you would do

```
$> uftp cp -R 0-1023 https://localhost:9000/rest/auth/TEST:/home/ ↩
    demo/test.data .
```

As an additional feature, you can use the additional "-p" flag, which will write also only the given range. For example

```
$> uftp cp -R 1024-2047-p https://localhost:9000/rest/auth/TEST:/ ↩
    home/demo/test.data .
```

will write bytes 1024-2047 of the remote file to the local file, starting at offset 1024. The local file will have length 2048.

The same thing works for remote files!

5.2.6 Encryption and compression

The cp command supports the "-E" and "-C" options, which enable data encryption and compression (during transfer) respectively. These work only if a single data stream is used.

Data encryption uses a symmetric algorithm, which nonetheless drastically lowers the performance.

Data compression uses the gzip algorithm.

Compression and encryption can be combined.

5.2.7 Resuming a failed transfer

If a copy command was terminated prematurely, it can be resumed using the "-R" option. If the "-R" option is present, the UFTP client will check if the target file exists, and will append only the missing data.

So if your inital copy operation

```
$> uftp cp -u username https://localhost:9000/rest/auth/TEST:/home/ ↩
    demo/test.data .
```

did not finish correctly, you can resume it with

```
$> uftp cp -R https://localhost:9000/rest/auth/TEST:/home/demo/test ↩
    .data .
```

5.2.8 Performance testing

For performance testing, you can use /dev/zero and /dev/null as data source / sink. For example to transfer 10 gigabytes of zeros from the remote server:

```
$> uftp cp -B 0-10G https://localhost:9000/rest/auth/TEST:/dev/zero ↩
    /dev/null
```

This can also be combined with the multi-connection option *-t*. To use two connections each transferring 5 gigabytes

```
$> uftp cp -B 0-10G -t 2 https://localhost:9000/rest/auth/TEST:/dev ↩
   /zero /dev/null
```

5.3 Synchronizing a file: the "sync" command

Currently, sync only supports single files, i.e. no directories or wildcards! The syntax is

```
$> uftp sync <master> <slave>
```

For example, to synchronize a local file with a remote "master" file:

```
$> uftp sync https://localhost:9000/rest/auth/TEST:/master.file  ↩
   local.file
```

To synchronize a remote file with a local "master" file:

```
$> uftp sync master.file https://localhost:9000/rest/auth/TEST:/ ↩
   remote.file
```

5.4 Data sharing

Data sharing enables users to create access to their datasets for other users via UFTP, even if those users do not have Unix-level access to the data.

Data sharing works as follows:

- when you share a file (or directory), the Auth server will store information about the path, the owner and the Unix user ID used to access the file in a database
- the targetted user can now access this file via the Auth server, and the Auth server will use the owner's Unix user ID to access the file.

By default, files will be shared for "anonymous" access. This will allow anyone who knows the sharing link to access the file using normal HTTP tools like *wget* or *curl*.

Shares can also be limited to certain users.

Depending on the type of share, access to the files is possible with the UFTP protocol or plain HTTPs.

Shares can be deleted by their owner, i.e. the user who created them.

Note
Not all UFTP installations support data sharing. You can check if a server has the sharing feature enabled by running "uftp info --server . . . "

5.4.1 Server URL

If not given via the --server argument, the URL of the Auth server will be taken from the environment variable UFTP_SHARE_URL

```
$> export UFTP_SHARE_URL=https://localhost:9000/rest/share/TEST
$> uftp share --list
```

5.4.2 Listing shares

```
$> uftp share --list --server https://localhost:9000/rest/share/ ↩
    TEST
```

The output will show both the files you have shared, as well as files that other users have shared with you.

5.4.3 Creating or updating a share

A share consists of a server-side path, (optional) write permissions and (optional) target user.

To share a file,

```
$> uftp share  \
      --server https://localhost:9000/rest/share/TEST  \
      /data/public/somefile.pdf
```

If you use a relative path, *uftp* will make it absolute.

```
$> pwd
/data/public/
$> uftp share somefile.pdf
```

will share the path "/data/public/somefile.pdf".

You can use the following options to modify the defaults:

- `--access <user-identifier>` to limit access to the specified user(s)
- `--write` for write acces
- `--delete` to delete a share

For example to share "/data/public/somefile.pdf" with the user "CN=User"

```
$> uftp share  \
      --server https://localhost:9000/rest/share/TEST  \
      --access "CN=User"  \
      /data/public/somefile.pdf
```

5.4.4 Deleting shares

To delete you need the path and the target user, which you can get via the "uftp share --list" command.

```
$> uftp share  \
      --delete  \
      --server https://localhost:9000/rest/share/TEST  \
      --access "CN=User"  \
      /data/public/somefile.pdf
```

5.4.5 Anonymous (http) access

For anonymous access via HTTP you need to use the correct URL. If you create (or list) shares, the *uftp* client will show the required links. You can download the file e.g. using *wget*.

5.4.6 Downloading/uploading using the UFTP protocol

To download a file that is shared with you, use the "get-share" command and the correct URL

```
$> uftp get-share https://localhost:9000/rest/share/TEST/auth:/data ↩
    /public/somefile.pdf
```

Currently this command does not support wildcards.

To upload a file to a location (file or directory) that has been shared with you, use the "put-share" command

```
$> uftp put-share data/*.pdf https://localhost:9000/rest/share/TEST ↩
    /auth:/data/public/
```

6 Using a proxy server (EXPERIMENTAL)

The uftp client has support for some types of FTP and HTTPs proxies.

This is configured via enviroment settings. I.e. in your shell you can define

FTP proxy

```
export UFTP_PROXY=proxy.yourorg.edu
export UFTP_PROXY_PORT=21
```

HTTP proxy

```
export UFTP_HTTP_PROXY=proxy.yourorg.edu
export UFTP_HTTP_PROXY_PORT=80
```

FTP proxying was tested with the "DeleGate/9.9.13" and "frox" proxies and requires UFTPD server version 2.8.1 or later to work.

If this does not work for you, or if you require support for a different type of proxy, please contact us via a support ticket or via email.

7 Troubleshooting

7.1 How can I get more detailed logging?

In the client's `conf` directory you'll find a `logging.properties` file that allows you to increase the log levels.

7.2 I get "Invalid server response 500" and "Exception. . . . Authentication failure"

Probably you gave a wrong username or password. Contact your site administrator if in doubt!

If using a password, make sure you give the "-P" flag.

7.3 I get "Invalid server response 405 Unable to connect to server for listing"

Check the remote URL that you use. Maybe you have a typo in the "/rest/auth/<servername>" part.

8 Getting support

UNICORE Website: http://www.unicore.eu

Support list: unicore-support@lists.sf.net

Developer's list: unicore-devel@lists.sf.net