

SCALE FOR PROJECT FT_P (/PROJECTS/FT_P)

Introduction

Please respect the following rules:

- Remain polite, courteous, respectful and constructive throughout the correction process. The well-being of the community depends on it.
- Identify with the person (or the group) graded the eventual dysfunctions of the work. Take the time to discuss and debate the problems you have identified.
- You must consider that there might be some difference in how your peers might have understood the project's instructions and the scope of its functionalities. Always keep an open mind and grade him/her as honestly as possible. The pedagogy is valid only and only if peer-evaluation is conducted seriously.


Guidelines

- Only grade the work that is in the student or group's GiT repository.
- Double-check that the GiT repository belongs to the student or the group. Ensure that the work is for the relevant project and also check that "git clone" is used in an empty folder.
- Check carefully that no malicious aliases was used to fool you and make you evaluate something other than the content of the official repository.
- To avoid any surprises, carefully check that both the correcting and the corrected students have reviewed the possible scripts used to facilitate the grading.
- If the correcting student has not completed that particular

project yet, it is mandatory for this student to read the entire subject prior to starting the defence.

- Use the flags available on this scale to signal an empty repository, non-functioning program, a norm error, cheating etc. In these cases, the grading is over and the final grade is 0 (or -42 in case of cheating). However, with the exception of cheating, you are encouraged to continue to discuss your work (even if you have not finished it) in order to identify any issues that may have caused this failure and avoid repeating the same mistake in the future.

Attachments

 Subject (https://cdn.intra.42.fr/pdf/pdf/5222/ft_p.en.pdf)

Preliminaries

Preliminary tests

Check firstly the following elements:

- There is something in the git repository.
- A valid author file
- The Makefile is present and compiles correctly the executables server and client
- No norm errors, Norminette is authoritative.
- No cheating (All functions are authorised, the student can explain the code, ...)

If an element of this list isn't respected, the grading ends.

Use the appropriate flag. You're allowed to debate some more about the project, but the grading will not be applied.

 Yes

 No

Server's testing

Simple test

The server launches, opens the port given as parameter and waits for a connection.

```
$> ./server 4242 & $> netstat -an | grep LISTEN | grep '4242 '
tcp4 0 0 0.0.0.0.4242 *.*
LISTEN $>
```

☒ Yes

☐ No

Less simple test

Connect to the server using nc:

```
$> nc 127.0.0.1 4242
```

Nothing crashes? nc is still connected?

☒ Yes

☐ No

Multiples connections

Can we connect a second client ?

☒ Yes

☐ No

Simple commands

For the following commands use the client in the repository or nc with the explanations from the student as protocol (the names of the functions are given as example. It is the functionality and not the syntax that is graded.)

Are these four commands functioning ?

- We can list a directory with ls
- We can change the current directory with cd
- We can break the connection with quit
- We can display the path to the current directory with pwd

☒ Yes

☐ No

get <file>

The commande get allows to download a file from the server to the client. Check with diff that the transfered file is strictly identical to the original.

☒ Yes

☐ No

put <file>

The commande put allows to upload a file from the client to the server. Check with diff that the transfered file is strictly identical to the original.

☒ Yes

☐ No

Advanced file transfer

Do a get of the server's binary AND a put of the client's binary. Both are functional?

☒ Yes

☐ No

Client's testing

Simple test

\$> ./client 127.0.0.1 4242

Is the client connecting?

☒ Yes

☐ No

Less simple test

\$> ./client localhost 4242

Is the client connecting?

☒ Yes

☐ No

Simple commands

This time we are only testing with the client and the server of the repository. Are all these commands functioning?

- We can list a directory with ls
- We can use options with ls (like ls -la for ex.)?
- We can exit with quit
- We can move with cd

☒ Yes

☐ No

get <file>

The commande get allows to download a file from the server to the client. Check with diff that the transfered file is strictly identical to the original.

☒ Yes

☐ No

put <file>

The commande put allows to upload a file from the client to the server. Check with diff that the transfered file is strictly identical to the original.

☒ Yes

☐ No

Bonus

IPv6 Support

The project can support both IPv4 and IPv6 simultaneously.

☒ Yes

☐ No

Respect of RFC

The project respects the FTP protocol defined by RFC (standard 9 or rfc 959).

☒ Yes

☐ No

Additional bonuses

If there are some additional bonuses, you can grade them here - you can add up to 5 bonuses. The bonuses must be 100% functional.

List of validated bonuses:

- lcd, lpwd and lls: these functions concern the ``local" filesystem and not the server.(All 3 of them are accounted as one bonus).
- mget: like get but "multiple", can contain some "*"
- mput: like put but "multiple", can contain some "*"
- directory mget
- directory mput
- login/password management
- rights control
- possibility to specify a different basic directory for each login
- '\n' conversion into '\r\n' (unix<->windows) of files ("bin" et "asc" modes: binary = no conversion, ascii = conversion of the transferred file)
- prompt
- completion during a get
- completion during a put
- mkdir
- rmdir
- unlink

Rate it from 0 (failed) through 5 (excellent)



Ratings

Don't forget to check the flag corresponding to the defense



Ok



Outstanding project



Empty work



Incomplete work



No author file



Invalid compilation



Norme



Cheat



Crash



Forbidden function

Conclusion

Leave a comment on this evaluation

Preview!!!