

(https://profile.intra.42.fr)


Remember that the quality of the defenses, hence the quality of the of the school on the labor market depends on you. The remote defences during the Covid crisis allows more flexibility so you can progress into your curriculum, but also brings more risks of cheat, injustice, laziness, that will harm everyone's skills development. We do count on your maturity and wisdom during these remote defenses for the benefits of the entire community.

SCALE FOR PROJECT PIPEX (/PROJECTS/PIPEX)

You should evaluate 1 student in this team



Git repository

git@vogsphere.42seoul.kr:vogsphere/intra-uuid-f718ebd4-382c-4e83-b98 

Introduction

Please comply with the following rules:

- Remain polite, courteous, respectful, and constructive throughout the evaluation process. The well-being of the community depends on it.
- Identify with the student or group whose work is evaluated the possible dysfunctions in their project. Take the time to discuss and debate the problems that may have been identified.
- You must consider that there might be some differences in how your peers might have understood the project's instructions and the scope of its

functionalities. Always keep an open mind and grade them as honestly as possible. The pedagogy is useful only and only if the peer-evaluation is done seriously.

Guidelines

- Only grade the work that was turned in the Git repository of the evaluated student or group.
- Double-check that the Git repository belongs to the student(s). Ensure that the project is the one expected. Also, check that "git clone" is used in an empty folder.
- Check carefully that no malicious aliases were used to fool you and make you evaluate something that is not the content of the official repository.
- To avoid any surprises and if applicable, review together any scripts used to facilitate the grading (scripts for testing or automation).
- If you have not completed the assignment you are going to evaluate, you have to read the entire subject before starting the evaluation process.
- Use the available flags to report an empty repository, a non-functioning program, a Norm error, cheating, and so forth.
In these cases, the evaluation process ends and the final grade is 0, or -42 in case of cheating. However, except for cheating, student are strongly encouraged to review together the work that was turned in, in order to identify any mistakes that shouldn't be repeated in the future.

Attachments

 subject.pdf (<https://cdn.intra.42.fr/pdf/pdf/24841/en.subject.pdf>)

Preliminaries

If cheating is suspected, the evaluation stops here. Use the "Cheat" flag to report it. Take this decision calmly, wisely, and please, use this button with caution.

Preliminary tests

If one of these points isn't valid, the correction stops.

- Empty git (= nothing in git repository).
- Norme error.
- Cheating (= -42).

If all of these points are valid, check yes and continues the correction.

☒ Yes

☐ No

General instructions

General instructions

- if the main program puts an error (Segfault, bus error, nonsense display, etc ...) use the flag crash!
- The Makefile compiles executable and has the required rules.
- The executable is named `pipex`.
- No prohibited function.

☒ Yes

☐ No

Mandatory part

The command `./pipex file1 cmd1 cmd2 file2` must behave like this command : `< file1 cmd1 | cmd2 > file2`

Error and arguments management

- The program takes 4 arguments, no more, no less (except for bonus part) and only in the right order.
- Error management is correct: existing files, files rights, the binary of the command exists etc.

If these points are respected, check `Yes` and continue the evaluation.
Otherwise, the evaluation is overuse Incomplete work or the appropriate flag.

✓ Yes

✗ No

The program

The program does what is requested, without displaying any additional information/steps against the shell command

Run your own tests et compare the results of shell exit and of shell output and that of the program.
If you haven't any idea, look at the subject examples.

If no error is detected, check `Yes` and continue.

✓ Yes

✗ No

Bonus

Evaluate the bonus part if, and only if, the mandatory part has been entirely and perfectly done, and the error management handles unexpected or bad usage. In case all the mandatory points were not passed during the defense, bonus points must be ignored.

multiple pipes

The program manages the usage of several pipes one after the other.
As for the mandatory part, test with shell commands then compare with program output.

If it's good only for two pipes in the same command but not for five,
the bonus is not counted.

☒ Yes☐ No

<< and >>

The program can copy the use of << and >>.

test multiple times something like :

CMD << STOP_VALUE | CMD1 >> file1

☒ Yes☐ No

Ratings

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

☒ Ok☐ Empty work☐ Norme☐ Cheat☐ Crash☐ Incomplete group☐ Leaks☐ Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation