

SCALE FOR PROJECT 42SH (/PROJECTS/42SH)

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 if the 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/2358/42sh.en.pdf>)

Mandatory part

Reminder : Remember that for the duration of the defence, no segfault, nor other unexpected, premature, uncontrolled or unexpected termination of the program, else the final grade is 0. Use the appropriate flag. This rule is active throughout the whole defence.

Author file

Check that the author file is at the root of the repository and formatted as explained in the subject. If not defence the is finished and final grade is 0.

 Yes

 No

Memory leaks

Throughout the defence, pay attention to the amount of memory used by 42sh (using the command top for example) in order to detect any anomalies and ensure that allocated memory is properly freed. If there is one memory leak (or more), the final grade is 0.

 Yes

 No

Fork and execve

"fork" et "execve" form the basis of a minimalist shell like the minishell. Therefore, those functions should be used in the code. Otherwise, it means that the instructions were

misunderstood.

Defence is finished and final grade is 0. The only way to succeed this project to use the list of authorized functions. There is no other solution.

Execute the following 4 tests:

- Run minishell, then run the following command "\$> foo". It must fail with a proper error message and then give back the prompt.

- Run the following command "\$> /bin/ls". ls must be properly executed and then give back the prompt.

- Run the following command "\$> /bin/ls -laF". ls must be properly executed with the -l, -a, -F flags and then give back the prompt.

- Run the following command "\$> /bin/ls -l -a -F". ls must be properly executed with the -l, -a, -F flags and then give back the prompt.

If at least one fails, no points will be awarded for this section. Move to the next one.

✓ Yes

✗ No

Builtins

In this section we'll evaluate the implementation of built-ins such as "exit", "echo" et "cd". A shell needs to include these basic features, even if it sounds prehistoric to you,

Execute the following 8 tests:

- Run minishell, then run the following command "\$> exit". The program must terminate properly and give back the parent's shell. Run the minishell again.

- Run a command such as "\$> echo "It works"". The message must be properly displayed.

- Run a command such as "\$> echo It works" (without the double quotes). The message must be properly displayed.

- Run a command such as "\$> cd /absolute/path/of/your/choice",

then run the following command "\$> /bin/pwd". /bin/pwd must confirm that the current folder was updated.

- Run a command such as "\$> cd relative/path/of/your/choice", then run the following command "\$> /bin/pwd". /bin/pwd must confirm that the current folder was updated.

- Run the following command "\$> cd", then run "\$> /bin/pwd". /bin/pwd must confirm that the current folder is the user's home folder.

- Run the following command "\$> cd -", then run "\$> /bin/pwd". /bin/pwd must confirm that the current folder is the folder relative/path/of/your/choice used before.

- Run the following command "\$> cd ~/path/of/your/choice", then run "\$> /bin/pwd". "\$> /bin/pwd". /bin/pwd must confirm that the current folder was updated.

If at least one fails, no points will be awarded for this section. Move to the next one.

✓ Yes

✗ No

Environment management

In this section we'll evaluate the implementation of specific built-ins such as "env", "setenv" et "unsetenv".

Execute the following 7 tests:

- Run the following command "\$> env". Environment variables must be displayed as key=value.

- Run a command such as "\$> setenv FOO bar" or "\$> setenv FOO=bar" depending on the implemented syntax. Then run the following command "\$> env". The environment must display a FOO variable with the value bar.

- Run the following command "\$> echo \$FOO". The value bar must be displayed.

- Run the following command "\$> /usr/bin/env". Minishell must send the appropriate environment to ran binaries. /usr/bin/env must display environment including FOO and its value bar.

- Run the following command "\$> unsetenv FOO". Then run "\$> env". The environment variable FOO must not be displayed anymore.

- Run the following command again "\$> unsetenv FOO". Then run "\$> env". Environment must not change.

- Run the following command again "\$> /usr/bin/env". /usr/bin/env must not display variable FOO anymore.()

If at least one fails, no points will be awarded for this section. Move to the next one.

 Yes

 No

PATH management

In this section we'll evaluate the implementation of PATH in your shell.

Execute the following 6 tests:

- Run the following command "\$> unsetenv PATH", then run "\$> setenv PATH "/bin:/usr/bin" or "\$> setenv "PATH=/bin:/usr/bin" depending on the implemented syntax. Then run the following command "\$> ls". /bin/ls must be properly executed.

- Run the following command "\$> emacs". /usr/bin/emacs must be properly executed.

- Run the following command "\$> unsetenv PATH", then run "\$> ls". It must fail.

- Run now the following command "\$> emacs". It must also fail.

- Run the following command "\$> /bin/ls". /bin/ls must be properly executed.

- Run the following command "\$> /usr/bin/emacs". /usr/bin/emacs must be properly executed.

If at least one fails, no points will be awarded for this section. Move to the next one.

✓ Yes

✗ No

Command line management

In this section we'll evaluate the command line management.

Execute the following 4 tests:

- Run an empty command "\$> ". The shell must do nothing and give back the prompt.

- Run a command made of just a single space "\$> ". The shell must do nothing and give back the prompt.

- Run a command made of spaces and tabulations. The shell must do nothing and give back the prompt.

- Run a command made of spaces and tabulations before and after its named and between its parameters such as "\$> /bin/l\$ /s -l -A". All those spaces and tabulations musn't interfere with the command's execution.

If at least one fails, no points will be awarded for this section. Move to the next one.

✓ Yes

✗ No

Pipes

In this section we'll evaluate pipe management.

Execute the following 4 tests:

- Run the following command "\$> ls | cat -e". The shell must display the content of the folder with a '\$' at the end of each line.

- Run the following command "\$> ls | sort | cat -e". The shell must display the sorted content of the folder with a '\$' at each line's end.

- Run the following command "\$> base64 /dev/urandom | head -c 1000 | grep 42 | wc -l | sed -e 's/1/Yes/g' -e 's/0/No/g'".

The shell must display "Yes" if the string "42" was found in the random characters, it'll display "No" otherwise.

If at least one fails, no points will be awarded for this section. Move to the next one.

 Yes

 No

Simple redirections

In this section we'll evaluate redirections.

Execute the following 3 tests:

- Run the following command "\$> echo "Testing redirections," > /tmp/test.txt" and check that the /tmp/test.txt file contains the string "Testing redirections".

- Run the following command "\$> echo "with multiple lines" >> /tmp/test.txt" and check that the /tmp/test.txt file contains the strings "Testing redirections," and "with multiple lines" on 2 lines.

- Run the following command "\$> wc -c < /tmp/test.txt" and check that the displayed value is 42.

If at least one fails, no points will be awarded for this section. Move to the next one.

 Yes

 No

Logical operators

In this section we'll evaluate logical operators implementation.

Execute the following 3 tests:

- Run the following command "\$> ls -l && ls" and check that ls is executed 2 times, one time using the long listing format the second time without.

- Run the following command "\$> ls something || ls" and check that the first ls fails and the second one is executed.

- Run the following command "\$> ls || ls something" and check

that the second `ls` isn't executed because the first one worked.

If at least one fails, no points will be awarded for this section. Move to the next one.

☒ Yes

☐ No

Several commands following each other

In this section we'll evaluate the management of several commands following each other with the `' ; '` separator.

Execute the following test:

- Run the following command `"$> ls -l ; touch ; ls -l"`. Both `ls` must be executed, the only difference being the "newfile" file.

If it fails, no points will be awarded for this section. Move to the next one.

☒ Yes

☐ No

A lil bit of everything

In this section we'll evaluate pipe, redirections, `' | '` all together.

Execute the following test:

- Run the following command `"$> mkdir test ; cd test ; ls -a ; ls | cat | wc -c > fifi ; cat fifi"`

The output must be:

```
..  
5
```

If it fails, no points will be awarded for this section. Move to the next one.

☒ Yes

☐ No

Optional part

We will look at your optional features if and only if your mandatory part is EXCELLENT. This means that you must complete the mandatory part, beginning to end, and your error management must be flawless, even in cases of twisted or bad usage. So if the mandatory part didn't score all the point during this defence bonuses will be totally IGNORED. There is no point in implementing extra exotic features in a shell that can't handle basic features properly.

Inhibitors

In this section we'll evaluate inhibitors implementation.

Execute the following 3 tests :

- Run the following command "\$> echo "" make sure that the shell wait for the closing double quotes, type some text ending with a double quotes. The text should be printed as if you used the echo command without double quotes.

- Run the following command "\$> echo " make sure that the shell wait for the closing single quote, type some text ending with a single quote. The text should be printed as if you used the echo command without single quote.

- Run the following command "\$> ls \" make sure that the shell wait for the rest of the command, type any flag you'd like to try out and press enter. The ls command should be executed with the flags as if you didn't use the \ inhibitor.

If at least one fails, no points will be awarded for this section. Move to the next one.

✓ Yes

✗ No

Heredoc

In this section we'll evaluate heredoc implementation.

Execute the following test:

- Run the following command "\$> cd /tmp; sort << EOF | cat -e > sorted_poem ; sed -e 's/Roses/Turnips/' < sorted_poem > better_poem; cd -; echo "I prefer turnips anyway" >> /tmp/better_poem; cat /tmp/better_poem" and type the following poem without the triple double quote but with the newlines:
""

Roses are red
Violets are blue
All my base are belong to you
I love you

"""

The output must be (without triple double quote):

"""

All my bases are belong to you\$

I love you\$

Turnips are red\$

Violets are blue\$

I prefer turnips anyway

"""

If it fails, no points will be awarded for this section. Move to the next one.

 Yes

 No

File descriptor aggregation

In this section we'll evaluate file descriptor aggregation.

Execute the following 3 tests:

- Run the following command "\$> rm nosuchfile 2>&-" making sure beforehand that there is indeed no file named 'nosuchfile' in the current folder. The error message shouldn't be displayed.

- Run the following command "\$> rm nosuchfile 2>&1 | cat -e" making sure beforehand that there is indeed no file named 'nosuchfile' in the current folder. The output must be: "rm: nosuchfile: No such file or directory\$"

- Run the following command "echo "No dollar character" 1>&2 | cat -e". The output must be: "No dollar character".

If at least one fails, no points will be awarded for this section. Move to the next one.

 Yes

 No

Line edition

In this section we'll evaluate simple line edition.

Execute the following 7 tests:

- It must be possible to move the cursor left or right in the

active command line using left and right arrows.

- It must be possible to edit the active command line where the cursor is located.

- It must be possible to jump at the beginning or at the end of the line using home and end keys.

- It must be possible to navigate through command line's history using up and down keys.

- It must be possible to move one word at a time left and right in the active command line using ctrl + left or right arrow.

- It must be possible to copy/paste part or all of the active command line with keyboard shortcuts.

- It must be possible to write and edit a command line on several lines at the same time.

If at least one fails, no points will be awarded for this section. Move to the next one.

 Yes

 No

Back quotes

In this section we'll evaluate back quotes implementation.

Execute the following test:

- Run the following command `"$> echo `ls -l`"` and make sure echo displays the result of the ls command and not the string "ls -l".

If it fails, no points will be awarded for this section. Move to the next one.

 Yes

 No

Globing

In this section we'll evaluate globing implementation.

Test the behavior of the shell regarding globing ("*", "?", "[",

"{}", etc) and make sure in the code that the function glob(3) isn't used.

If at least one of your tests fails, no points will be awarded for this section. Move to the next one.

☒ Yes

☐ No

Sub-shell

In this section we'll evaluate sub-shell implementation.

Check out that when using sub shells using "()" the commands are properly executed without issues or problems.

If at least one of your tests fails, no points will be awarded for this section. Move to the next one.

☒ Yes

☐ No

Local variables

In this section we'll evaluate local variables implementation.

Check out that when local variables and the unset and export builtins are implemented.

If at least one of your tests fails, no points will be awarded for this section. Move to the next one.

☒ Yes

☐ No

History

In this section we'll evaluate history implementation.

Check out that an history system is implemented. It doesn't necessary require the use of the command history, but needs to have similar features such as being able to recall commands from the history, clear the history etc.

If at least one of your tests fails, no points will be awarded for this section. Move to the next one.

✓ Yes

✗ No

Files manipulation

In this section we'll evaluate file manipulation implementation.

Check out that file descriptors work as expected and that the read builtin is implemented with all its features.

If at least one of your tests fails, no points will be awarded for this section. Move to the next one.

✓ Yes

✗ No

Files manipulation

In this section we'll evaluate auto-completion implementation.

Check out that when you type the beginning of a command or a path there is a way, like pressing the TAB key for example, to complete it.

If at least one of your tests fails, no points will be awarded for this section. Move to the next one.

✓ Yes

✗ No

Job control

In this section we'll evaluate job control implementation.

Check out that it's possible to use the following builtins and all their features:

- job
- fg
- bg

As well as the "&" operator.

If at least one of your tests fails, no points will be awarded for this section. Move to the next one.

✓ Yes

✗ No

Bonus

Reminder : Remember that for the duration of the defence, no segfault, nor other unexpected, premature, uncontrolled or unexpected termination of the program, else the final grade is 0. Use the appropriate flag. This rule is active throughout the whole defence. We will look at your bonuses if and only if your mandatory part and the optional part are EXCELLENT. This means that you must complete both parts, beginning to end, and your error management must be flawless, even in cases of twisted or bad usage. So if the mandatory part and the optional part didn't score all the point during this defence bonuses will be totally IGNORED.

Shell script

In this section we'll evaluate shell script implementation.

Check out that shell script is fonctionnal.

If at least one of your tests fails, no points will be awarded for this section. Move to the next one.

☒ Yes

☐ No

Additional features

If the 42sh has additional features you can grade them in this section.
The additional features have to be useful and well implemented without faulting the stability of the shell.

If at least one of your tests fails, no points will be awarded for this section.

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

☐ Incomplete group

☐ Forbidden function

Conclusion

Leave a comment on this evaluation

Preview!!!

General term of use of the site
(<https://signin.intra.42.fr/legal/terms/6>)

Privacy policy
(<https://signin.intra.42.fr/legal/terms/5>)

Legal notices
(<https://signin.intra.42.fr/legal/terms/3>)

Declaration on the use of cookies
(<https://signin.intra.42.fr/legal/terms/2>)

Terms of use for video surveillance
(<https://signin.intra.42.fr/legal/terms/1>)

f
(<https://signin.intra.42.fr/legal/terms/7>)