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 21SH (/PROJECTS/21SH)

You should evaluate 1 student in this team

★

Git repository

```
git@vogsphere-v2.hive.fi:vogsphere/intra-uuid-7b7a0afe-cf97-4843-8df
```

Introduction

Please respect 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 evaluated person (or group) the potential malfunctions of the work. Take the time to discuss and debate the problems you have 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 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 were 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 evaluator and the evaluated students have reviewed the possible scripts used to facilitate the grading.

- If the evaluator has not completed that particular project yet, it is mandatory for this student to read the entire subject prior to starting the defense.

- 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/6606/21sh.en.pdf)

Mandatory part

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

Author file

Check that the author file is at the root of the repository and formatted as explained in the subject. If not, the evaluation is over and the final grade is **0**.

Yes No

Memory leaks

Throughout the evaluation, pay attention to the amount of memory used by **21sh** (using the commands `top` or `leaks` 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

The `fork` and `execve` functions form the basis of a minimalist shell, like the minishell. Therefore, those functions must be used in the code. Otherwise, it means that the instructions were misunderstood - the evaluation is over and the final grade is **0**.

Execute the following 4 tests. If at least one fails, no points will be awarded for this section and you should move to the next one.

- Run **21sh**, 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` and `-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` and `-F` flags and then give back the prompt.

☑ Yes ☒ No

Builtins

In this section, we'll evaluate the implementation of the `exit`, `echo` and `cd` built-ins.

If a single one of these built-ins is not implemented or does not work properly, no points should be given for this section - you can move to the next one.

- Run **21sh**, then run the following command:

```
$> exit
```

The program must terminate properly and give back the parent's shell. Run **21sh** 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` must confirm that the current folder was updated.

☑ Yes ☒ No

Environment management

In this section we'll evaluate the implementation of the `env`, `setenv` and `unsetenv` built-ins.

If a single one of these built-ins is not implemented or does not work properly, no points should be given for this section - you can move to the next one.

- Run the following command:

```
$> env
```

.
The 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 `$> env`. The environment must display a `FOO` variable with the value `bar`.

- Run the following command:

```
$> /usr/bin/env
```

21sh must send the appropriate environment to run binaries. It must display the 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`. The environment must not change.

- Run the following command again:

```
$> /usr/bin/env
```

`/usr/bin/env` must not display variable `FOO` anymore.

☑ Yes ☒ No

PATH management

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

Execute the following 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 a single one of these tests fails, no points should be given for this section - you can move to the next one.

Yes No

Command line management

In this section, we'll evaluate the command line management. Execute the following 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 containing spaces and tabulations before and after its name, as well as between its parameters, for example:

```
$>      /bin/ls       -l      -A
```

All those spaces and tabulations must not interfere with the command's execution.

If a single one of these tests fails, no points should be given for this section - you can move to the next one.

Yes No

Signal

In this section, we'll evaluate signal management and more specifically `Ctrl-C` . Execute the following tests.

- Instead of typing a command press `Ctrl-C` . The shell must just give back the prompt.
- Type a random command but instead of running it, press `Ctrl-C` . The program must give back an empty prompt.
- Run the following command:

```
$> cat
```

When cat waits for an input on the standard input, press `Ctrl-C` . The program must kill `cat` 's process and give back the prompt.

If a single one of these tests fails, no points should be given for this section - you can move to the next one.

Yes No

Pipes

In this section, we'll evaluate pipe management. Execute the following 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 the end of each line.

- Run the following command:

```
$> base64 /dev/urandom | head -c 1000 | grep 42 | wc -l | sed -e 's/1/Yes/g' -
```

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

If a single one of these tests fails, no points should be given for this section - you can move to the next one.

Yes No

Redirections

In this section, we'll evaluate redirections. Execute the following tests.

- Run the following command:

```
$> echo "Testing redirections," > /tmp/test.txt
```

Check that the `/tmp/test.txt` file contains the string "Testing redirections,".

- Run the following command:

```
$> echo "with multiple lines" >> /tmp/test.txt
```

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
```

Check that the displayed value is 42.

- Run the following command:

```
$> cat -e << EOF
```

Then, type the following poem with the newlines:

```
Roses are red
Violets are blue
All my base are belong to you
And so are you
```

Then press `CTRL+D` to stop the input. The command's output must be exactly as follow:

```
Roses are red$
Violets are blue$
All my base are belong to you$
And so are you$
```

- Run the following command:

```
$> cat -e << EOF >> /tmp/test.txt
```

Write the last poem once more. Check that the `/tmp/test.txt` file contains the following 6 lines:

```
Testing redirections,
with multiple lines
Roses are red$
Violets are blue$
All my base are belong to you$
And so are you$
```

If a single one of these tests fails, no points should be given for this section - you can move to the next one.

☑ Yes ☒ No

Several commands following each other

In this section, we'll evaluate the management of several commands following one another with the `;` separator. Execute the following test.

Run the following command

```
$> ls -1; touch newfile  ; ls -1
```

Both `ls` must be executed, the only difference being the `newfile` file.

If it fails, no points should be given for this section - you can move to the next one.

☑ Yes ☒ No

A lil' bit of everything

In this section, we'll evaluate pipe, redirections, `;` altogether.

Execute the following tests:

- Run the following command:

```
$> mkdir test ; cd test ; ls -a ; ls | cat | wc -c > fifi ; cat fifi
```

The output must be:

```
. ..
       5
```

- Run the following command:

```
$> cd /tmp; sort << EOF | cat -e > sorted_poem ; sed -e 's/Roses/Turnips/' < s
```

Type the following poem with the newlines:

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

The output must be as following:

```
All my bases are belong to you$
I love you$
Turnips are red$
Violets are blue$
I prefer turnips anyway
```

If a single one of these tests fails, no points should be given for this section - you can move to the next one.

☑ Yes ☒ No

File descriptor aggregation

In this section, we'll evaluate file descriptor aggregation. Execute the following.

- Run the following command:

```
$> rm nosuchfile 2>&-
```

Make 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
```

Make 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 a single one of these tests fails, no points should be given for this section - you can move to the next one.

☑ Yes ☒ No

Simple line edition

In this section, we'll evaluate the simple line edition. Execute the following tests.

- It must be possible to move the cursor left or right in the active command line using the 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 the command line's history using up and down keys.

If a single one of these tests fails, no points should be given for this section - you can move to the next one.

Yes No

Advanced line edition

In this section, we'll evaluate the advanced line edition. Execute the following tests.

- 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. (1 point)
- It must be possible to copy/paste part or all of the active command line with keyboard shortcuts. (2 points)
- It must be possible to write and edit a command line on several lines at the same time. (2 points)

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

---

ctrl+D and ctrl+C

In this section, we'll evaluate `ctrl+D` and `ctrl+C` management. Execute the following tests.

- Press `ctrl+D` when the active command line is empty. The shell must exit properly.
- Press `ctrl+D` when the active command line isn't empty. Nothing should happen.
- Run the command `$> cat` , type a few characters then press `ctrl+D` 2 times. It must first output the characters typed, then give back the prompt.
- Press `ctrl+C` when the active command line is empty and when it isn't empty. In both cases. the shell must give the prompt back.
- Run the command `$> cat` then press `ctrl+C` . The shell must kill `cat` 's proccess and give back the prompt.

If a single one of these tests fails, no points should be given for this section - you can move to the next one.

Yes No

Quotes management

In this section. we'll evaluate quotes management. Execute the following test.

Run the command

```
$> echo "
```

Press enter. The shell must start a new line and wait for the end of the command. Type some more lines, and close the double quote. The shell must get the whole command and execute it normally.

For example:

```
$> echo "
*>Roses are red
*>Violets are blue
*>All my base are belong to you
*>I love you
*>"

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

$>
```

If the test fails, no points should be given for this section.

☑ Yes ☒ No

Bonus

You will look at the bonuses if and only if the mandatory part is EXCELLENT. This means that the mandatory part must be validated in its entirety (the evaluated group got all the points) and the error management must be flawless, even in cases of twisted or bad usage.

Other features

You can give points for other relevant and interesting bonuses up to 5.

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 Leaks Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation