(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 bene fits of the entire community.

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

You should evaluate 4 students in this team

Git repository

Introduction

We ask you for the proper conduct of this evaluation of
comply with the following rules:

You have been using SHELL for many months, you know how to
he has to react.

Be imaginative about error testing and remember that finding
a SEGFAULT means for the corrected that their work has not
been quite thorough. For the proofreader, this means that he must
make sure to discuss what causes this SEGFAULT with your employees
corrected.

In case of segfault, nevertheless continue the defense for
to discuss the subject (without applying the scale, since you
will of course have selected the CRASH tab below).

- Accept that there may sometimes be differences
interpretation on the subject's requests or the scope of the
functionalities. Remain open-minded about the vision of
the other (is he or she right or wrong?), and write down the most
honestly possible. The pedagogy of 42 only makes sense if
peer evaluation is done seriously.

Guidelines

- You only need to evaluate what is on the GiT rendering deposit
of the student or group.

First correct the mandatory part. If and only if this
mandatory part is PERFECT (complete and

indestructible!), switch to the modular part. Keep in mind that the shell must be intuitive, but that the corrected ones remain free to the implementation of certain details. Their reference shell is not necessarily the same as yours.

The correction can be long if the modular part is correct provided. The 42sh is not a trivial project, take the time to make it to scrutinize your classmates' work.

- Any script that is intended to facilitate the evaluation provided by one of the twoparties must be rigorously verified by the other party to avoid unpleasant surprises.

- If the student proofreader has not yet done thisproject, it is mandatory for this student to read the subject in its entirety before starting this defense.

- Use the flags available on this scale to report a made empty, non-functional, a fault of standard, a case of cheating, etc. In this case, the evaluation is completed and the final score is 0 (or -42 in the special case of cheating). However, except in the case of of cheating, you are encouraged to continue to share around the work done (or not done) to identify the problems that caused this situation and avoid them for the next render.

- The return value can be retrieved with "echo ${?}".

- The user's prompt will be represented by "$>" in all tests in the scale.
You will therefore have to adapt your correction according to the prompt of the group being evaluated.

- The parts contained in the placeholders "{{ ... }}" indicate a partwhich is likely to vary according to the corrections.
Instead, we tell you what is likely to be displayed.
Therefore, it is your responsibility to ensure that the party displayed is in line with the placeholder's theme.

Attachments

 files needed for correction (/uploads/document/document/1549/42sh.txts.tgz)

 Subject (https://cdn.intra.42.fr/pdf/pdf/2362/42sh.en.pdf)

Mandatory part

Reminder: if at any time the program does not react correctly (bus error, segfault, etc...), or if you detect a memory leak, the defense is over and the note is 0. Remember to use the corresponding flags when this is necessary. This command is active throughout the defense.
Author file

First check the following items:

- There is a rendering (in the git repository)
- Valid author file
- The Makefile is present and compiles the 42sh executable well
- No fault of standard, the Norminette being authoritative- No cheating (unauthorized functions, students must be able to explain their code,...)

If an element does not conform to the subject, the rating stops there. You are encouraged to continue to discuss of the project, but the scale is not applied

 Yes No

Leaks

Throughout the duration of the defense, keep an eye on the possible 42sh memory leaks (via this command in another terminal by example "while true; do leaks 42sh; sleep 1; clear; done". If leaks reports a memory leak, the project score is 0.

 Yes No

Prerequisites from minishell

We will evaluate the prerequisites of the minishell.
If at least one of these tests fails, then the entire section is failed and the correction stops. Perform the following tests:

- Execute an empty command "$>". The shell must do nothing anddisplay the prompt again.

- Execute a command with only one "$> " space.
The shell must do nothing and display the prompt again.

- Execute a command composed only of spaces andtabulations. The shell must do nothing and display the prompt again.

- Execute a command with several spaces and tabsbefore the binary name, between each argument passed to the binary, and after the last argument. These spaces and unnecessary tabulations must not disrupt the execution of the order.

- Test a command (and/or built-in) with non-existent options.Check that the return of the order is not 0.

- Execute the following command and check that the display is correct:

```
$> doesnotexist {{ Error message indicating that the command does not
exist/is not found }}
$> echo ${?}
{{ Return code different from 0 }}
```

- Execute the following command and check that the display is correct:

```
$> /sbin/yubikey_shell {{ Error message indicating that the command cannot be
executed due to insufficient permission }} $> echo ${?}
{{ Return code different from 0 }}
```

- Execute the following command and check that the display is correct:

```
$> /bin/ls {{ Output of the
"ls" command }}
$> echo ${?}
0
```

- Execute the following command and check that the display is correct:

```
$> ls {{ Output of the "ls"
command }}
$> echo ${?}
0
```

- Execute the following command and check that the display is correct:

```
$> /bin/ls -alF {{ Output of the "ls" command with the
"alF" arguments }}
$> echo ${?}
0
```

- Execute the following command and check that the display is correct:

```
$> /bin/ls -l -a -F {{ Output of the "ls" command with
the "alF" arguments }}
$> echo ${?}
0
```

Don't be satisfied with the correction tests, do your own tests in addition. For example, you can test redirections inside pipes works.

Yes No

Prerequisites from 21sh

We will evaluate the prerequisites of the 21sh.

If at least one of these tests fails, then the entire section is failed and the correction stops. Perform the following tests:

- Check that it is possible to move around in the command linevia various shortcuts and edit it at the cursor location.

- Execute the following command and check that the display is correct:

```
$> ls > /tmp/ftsh_ls_out /
$> cat /tmp/ftsh_ls_out
{{ Output of the "ls" command on the system root }}
```

- Execute the following command and check that the display is correct:

```
$> < /tmp/ftsh_ls_out cat -e >> /tmp/ftsh_ls_out
$> cat /tmp/ftsh_ls_out {{ 2 listings of the root must appear and the second must
have a $ at the end of each line }}
```

- Perform the following commands and check that the display is compliant:

```
$> echo 1 >out >&2 2>err
1
$> echo 2 >out 2>err
$> cat err
$> cat out
2
$>
```

- Perform the following commands and check that the display is compliant:

```
$> echo non-standard fd > dup_fd
$> cat 4 non-standard fd$
$>
```

- Execute the following command and check that the display is correct:

```
$> cat <&4
{{ Error message indicating that the file descriptor is invalid }}
$>
```

- Perform the following commands and check that the display is compliant:

```
$> echo abc >redir_one_to_all
$> cat 9 abc$
$>
```

- Execute the following command and check that the display is correct:

```
$> cat <&- abc
```

```
$>
```

- Perform the following commands and check that the display is compliant:

```
$> ls doesnotexist . 2>&1 >/dev/null ls:
doesnotexist: No such file or directory $>
ls doesnotexist . >/dev/null 2>&1
$>
```

- Execute the following command and check that the display is correct:

```
$> ls | sort -rn | cat -e {{ Content of the current folder, sorted, with a '$'
at the end of each line }}
$>
```

- Execute the following command and check that the display is correct:

```
$> base64 < /dev/urandom | head -c 1000 | grep 42 | wc -l | sed -e s/1/Yes/g -
e s/0/No/g {{ Displays "Yes" or "No" randomly }} $> ps a | grep 'base64' |
grep -v 'grep'
$>
```

- Execute the following command and check that the display is correct:

```
$> ls -1; touch test_file; ls -1 {{ Display of 2 'ls'. An additional file,
"test_file", must appear in the second output
}}
$>
```

- Execute the following command and check that the display is correct:

```
$> exit 1 | exit 2 | exit 3; echo "stayin'
alive" stayin' alive $>
```

Check that the 42sh has not finished and that the prompt is
available.

- Execute the following command and check that the display is correct:

```
$> echo out >&-; echo out2 {{ Optional error message indicating that it is
impossible to write to stdout }} out2 $> echo out >&- | echo out2 {{ Optional
error message indicating that it is impossible to write to stdout }} out2 $>
echo out >&- && echo out2 {{ Optional error message indicating that it is
impossible to write to stdout }}
$> echo out >&- || echo out2 {{ Optional error message indicating that it is
impossible to write to stdout }} out2
```

- Execute the following command and check that the display is correct:

```
$> cat << END
heredoc> hello world
heredoc> and good
heredoc> morning!
heredoc> END
hello world
and good
morning! $>
```

- Execute the following command and check that the display is correct:

```
$> cat << EO\
> F heredoc>
hi heredoc>
EOF hi
```

- Execute the following command and check that the display is correct:

```
$> cat > /tmp/heredoc-append << FIN
heredoc> abc heredoc> FIN $> cat -e >>
/tmp/heredoc-append << FIN heredoc>
def heredoc> ghi
heredoc> FIN $> cat
/tmp/heredoc-append abc
def$ ghi$ $>
```

- Execute the following command and check that the display is correct:

```
$> (cat < heredoc> abd
heredoc> abc heredoc>
abb heredoc> EOF abb$
abc$ abd$ $>
```

Don't be satisfied with the correction tests, do your own tests in addition. For example, you can test redirections inside pipes works.

 Yes No

Built-ins

We will evaluate the implementation of the builtins "exit", "echo", "cd" and "type".

If at least one of these tests fails, then the entire section is
failed and the correction stops. Perform the following tests:

- Perform the following commands and check that the display is compliant:

```
$> echo abc; exit; echo def
abc $> echo ${?}
0
```

- Execute the following command and check that the display is correct:

```
$> exit 999999999999999999999999999999999999999999
```

Check that the 42sh's behavior is consistent and that no crashes occur.
or undetermined behaviour has occurred.

- Perform the following commands and check that the display is compliant:

```
$> exit abc {{ Error message indicating that a numerical value
is expected }}
$> echo ${?}
{{ Return code different from 0 }}
```

- Perform the following commands and check that the display is compliant:

```
$> exit 1 2 3 {{ Error message indicating too
many arguments }}
$> echo ${?}
{{ Return code different from 0 }}
```

- Perform the following commands and check that the display is compliant:

```
$> cd {{ /absolute/path/of/your/choice }}
$> pwd {{
/absolute/path/of/your/choice }}
$>
```

- Perform the following commands and check that the display is compliant:

```
$> cd relative/path/of/your/choice
$> pwd {{
relative/path/of/your/choice }}
$>
```

- Perform the following commands and check that the display is compliant:

```
$> cd /tmp
```

```
$> /bin/pwd
/tmp
$> cd
$> /bin/pwd
/Users/{{login_session}}
$>
```

- Perform the following commands and check that the display is compliant:

```
$> cd /tmp
$> pwd
/tmp
$> cd /bin
$> pwd
/bin
$> cd -
$> pwd
/tmp
$>
```

- Perform the following commands and check that the display is compliant:

```
$> cd -L /tmp; cd -P ..
$> pwd
/private
$>
```

- Execute the following command and check that the display is correct:

```
$> type type ls {{ Message indicating that "type" is a builtin and "ls" a command
with its path }}
$>
```

Don't be satisfied with the correction tests, do your own tests in addition. For example, you can test the right behaviour the built-in "cd" if the variable "CDPATH" is present in the environment.

Yes No

Logical operators

We will evaluate the implementation of logical operators.

If at least one of these tests fails, then the entire section is failed and the correction stops. Perform the following tests:

- Execute the following command and check that the display is correct:

```
$> ls -l && ls {{ Display of the "ls" command twice with
different parameters }}
$>
```

- Execute the following command and check that the display is correct:

```
$> ls doesnotexist || echo "Notice me senpai"
ls: doesnotexist: No such file or directory
Notice me senpai
$> echo ${?}
0
$>
```

- Perform the following commands and check that the display is compliant:

```
$> echo 'No error' || echo 'You cant see me'
No error
$> echo ${?}
0
$>
```

- Perform the following commands and check that the display is compliant:

```
$> false && echo foo || echo
bar bar $> true || echo foo &&
echo bar bar $>
```

Don't be satisfied with the correction tests, do your own tests in addition.

 Yes No

Environment management

We will evaluate the support of internal and external variables, as well as the implementation of the builtins "set", "export" and "unset".

If at least one of these tests fails, then the entire section is failed and the correction stops. Perform the following tests:

- Execute the following command and check that the display is correct:

```
$> a=hello b=world; b=42 echo ${a}_${b} && echo
${b} hello_world world $>
```

- Perform the following commands and check that the display is compliant:

```
$> directory=/ ls_opt=-atr
$> ls ${ls_opt} ${directory} {{ Output of the "ls -atr"
command on the system root }}
$>
```

- Execute the following command and check that the display is correct:

```
$> echo ${empty}|cat -e
$
$>
```

- Perform the following commands and check that the display is compliant:

```
$> set {{ All internal shell and environment
variables }} $> set | grep -E '(a|b)=' a=hello
b=world $>
```

- Execute the following command and check that the display is correct:

```
$> env
{{ All environment variables only }}
$> env | grep -E '(a|b)='
$>
```

- Perform the following commands and check that the display is compliant:

```
$> export b
$> printenv
b world $>
```

- Perform the following commands and check that the display is compliant:

```
$> ONESHOT= env | grep ONESHOT
ONESHOT=
$> env | grep ONESHOT
$>
```

- Perform the following commands and check that the display is compliant:

```
$> unset a b
```

```
$> env | grep -E '(a|b)='
$> set | grep -E '(a|b)='
$>
```

- Perform the following commands and check that the display is compliant:

```
$> unset PATH
$> PATH=/bin:/usr/bin
$> mkdir testdir
$> echo ${?}
0
$> ls -1 | grep testdir
testdir $>
```

- Perform the following commands and check that the display is compliant:

```
$> true; echo ${?}; false; echo ${?}
0
1
$>
```

Don't be satisfied with the correction tests, do your own tests in addition. For example, you can test the built-in options "export".

 Yes No

Job control

We will evaluate the implementation of job control.

Job control allows to control processes in an interactive way by allowing orders to be placed in the background, to stop them and to take them back into the foreground.

If at least one of these tests fails, then the entire section is failed and the correction stops. Perform the following tests:

- Perform the following commands and check that the display is compliant:

```
$> mkfifo fifo $> ls -lR
/usr >fifo 2>&1 &
$> jobs {{ Message indicating that the "ls" command is being
executed }}
$>
```

- Perform the following commands and check that the display is compliant:

```
$> emacs -nw &
$> emacs -nw &
$> emacs -nw &
$> emacs -nw &
$> emacs -nw &
$> jobs {{ Message indicating that 5 instances of emacs are stopped in the
background }}
$>
```

- Execute the following command and check that the display is correct:

```
$> fg %{{ one of the emacs job number }}
```

The Emacs process must come back to the foreground and be functional.
Repeat as many times as necessary to bring back all the "emacs" processes.

- Perform the following commands and check that the display is compliant:

```
$> jobs
{{ The "ls" command launched above must be the only one left }}
$> cat -e {{ Output of the "ls" command. Do not wait until the end, cut the
display with CTRL-C }} $> jobs
$>
```

- Perform the following commands and check that the display is compliant:

```
$> ls -Rl / 2>&1 {{ Display of the "ls" command. Don't wait until the
end, press CTRL-Z }}
{{ Message indicating that the order is suspended }}
$> jobs {{ Message indicating that the order is
suspended }}
$>
```

- Perform the following commands and check that the display is compliant:

```
$> ps a | grep "ls -Rl /" | grep -v 'grep' | cut -d ' ' -f 2
{{ PID of the command "ls" from the previous test }} $>
kill {{ PID of the command "ls" from the previous test }}
$> jobs {{ Message indicating that the order has
ended }}
$>
```

- Perform the following commands and check that the display is compliant:

```
$> jobs $> ps a | grep "ls -Rl /" | grep
-v 'grep'
$>
```

Don't be satisfied with the correction tests, do your own tests in addition. For example, you can test the built-in "bg".

 Yes No

signals

We will evaluate the signal management.

If at least one of these tests fails, then the entire section is
failed and the correction stops. Perform the following tests:

- Check that the shell correctly manages the signals emitted by its children.To do this, you can use the following command:

```
$> python -c 'import os, signal;os.kill(os.getpid(), signal.SIGSEGV)'
{{ Message indicating the received signal }}
$>
```

Replace "SIGSEGV" with the signal you want to send.
Test all signals!
The shell must in no way leave if one of its children
ends by a signal, even if it's a "SIGKILL".

- Check that "CTRL-C" in an empty prompt and with a commandre-displays an empty prompt.

- Execute the command:

```
$> cat
```

Then press "CTRL-\".
The "cat" command must end with a message indicating the signal
received and the prompt be available.

Don't be satisfied with the correction tests, do your own
tests in addition.

Yes No

Optional part

Reminder: you should only evaluate the modular part if the mandatory part is PERFECT. There is no interest in develop exotic features on a shell that does not perfectly provide the basic functionalities! The tests for the following sections are in separate files. You must perform all tests. If at least one of these tests fails, then the whole section is failed and you move on to the next one. Each file contains a transcript of a shell session. You must reproduce the commands listed in the file and make sure that the display matches to what is expected. Be careful with the prompts in the files: - "$> " represents the normal prompt, waiting for a command - "> " represents an incomplete command waiting for a user input ((heredoc, quote, dquote...)

Inhibitors

In this section we will evaluate the presence and correctness of inhibitors """ (double quote), "'" (single quote) and "\" (backslash).

A "quote>" prompt indicates that the shell is waiting for an additional entry to complete the current one.

The tests for this section can be found in the file "42sh.quoting.txt".

Don't be satisfied with the correction tests, do your own tests in addition.

For example, you can test these commands and check that the shell reacts in the same way as the reference shell chosen by the group:

```
$> echo foo\
$> echo "\\'abcd\\'"
$> echo \'
```

 Yes No

Pattern Matching

In this section we will evaluate the proper functioning of globing ("*", "?", "[]", "-", "!").
Look in the sources for the implementation of globbing, the glob(3) function should not be used.

The tests for this section can be found in the file "42sh.pattern_matching.txt".

Do your own tests too!
For example, you can test how the shell behaves if an element of the pattern is escaped (with "\\") or if the pattern is between inhibitors ("'")

 Yes No

Additional Expansion

Test the presence and proper functioning of tilde expansions and of the parameters.

The tests for this section can be found in the file "42sh.expansions.txt".

Don't be satisfied with the correction tests, do your own tests in addition. There are many possible tests for expansion. Be inventive!

Yes No

Grouped controls and sub-shells

Test the presence and proper functioning of the sub-shells and grouped controls:

The tests for this section can be found in the file "42sh.grouped_commands.txt".

Don't be satisfied with the correction tests, do your own tests in addition.
Also test syntax errors, such as:

```
$> ()
$> (echo a|)
$> (; echo b)
$> (echo c; ())
```

Yes No

Substitution of commands

Test the presence and proper functioning of command substitution

The tests for this section can be found in the file "42sh.command_sub.txt".

Don't be satisfied with the correction tests, do your own tests in addition. There are many possible tests for the command substitution.
Be inventive!

Yes No

Arithmetic Expansion

Test the presence and proper functioning of arithmetic expansions

The tests for this section can be found in the file "42sh.exp_arithm.txt".

Also test that the expansion is done between double quote ("""") and not between simple quote (""''). Test inconsistent values, such as that "99999999999999999999999999999999 + 1"

 Yes No

Substitution of processus

Test the presence and proper functioning of process substitution

The tests for this section can be found in the file "42sh.process_sub.txt".

Don't be satisfied with the correction tests, do your own
tests in addition. There are many possible tests for
substitution of processes.
Be inventive!

 Yes No

History

Test the presence and proper functioning of the history.

The tests for this section can be found in the file "42sh.history.txt".

In addition, perform the following tests:

- Use the command "fc -l" to get the commands historywith their indexes. Then execute a command with the expansion "!". Check that the correct command is being executed

- Same as above but this time with the expansion "!-".
Check that the correct command is being executed

- Check that the commands are recorded in a file. Close and restart the shell. Does the history of the old session is accessible in the new shell?

- Execute the command:

```
$> fc -e vim -1 -10
```

Check that the last 10 commands are present in vim.
Edit them if you wish, save and close vim.
The commands present in vim at closing must be executed.

- Check that the incremental search, via a shortcut CTRL-R(or another) works.

Don't be satisfied with the correction tests, do your own tests
in addition. There are many possible tests for historical
information and its expansion.
Be inventive!

 Yes No

Contextual dynamic completion

Test the presence and proper functioning of the dynamic completion.

If at least one of these tests fails, then the entire section is
failed and the correction stops. Perform the following tests:
- Check that the completion of commands in the "PATH" works

- Same for builtins

- Check that the completion is contextual. If you have
"ls /sbin/" on the command line and press TAB
(or any other key responsible for completion)
then only the files in the "/sbin" directory should
appear.

- Execute the commands:

```
$> abc=def
$> echo ${a
```

Check that the completion offers you the variable "abc".

- Execute the commands:

```
$> unset a
$> echo ${a
```

Check that the completion no longer offers you the variable "abc".

Don't be satisfied with the correction tests, do your own
tests in addition. There are many possible tests for
completion dynamic.
Be inventive!

 Yes No

Editing mode Vi/Readline

Test the presence and proper functioning of Vi editing
modes and Readline.

If at least one of these tests fails, then the entire section is
failed and the correction stops. Perform the following tests:

- Check that it is possible to change the editing mode with
thecommand "set -o vi" or "set -o readline"

- Check that all the shortcuts mentioned in the subject are
workingcorrectly for both modes.

Don't be satisfied with the correction tests, do your own tests in addition. There are many possible tests for editing of line. Be inventive!

 Yes No

Built-ins alias/unalias

Test the presence and proper functioning of aliases

The tests for this section can be found in the file "42sh.alias.txt".

Don't be satisfied with the correction tests, do your own tests in addition. There are a lot of possible tests for aliases.
Be inventive!
For example, you can test invalid alias names, such as "=", "-" or "/".
The shell must display an error.

 Yes No

Hash table

Test the presence and proper functioning of the hash table and of the builtin "hash"

The tests for this section can be found in the file "42sh.hash.txt".

Don't be satisfied with the correction tests, do your own tests in addition. There are many possible tests for the table of hashing and the builtin "hash". Be inventive!

 Yes No

Built-in test

Test the presence and proper functioning of the "test" builtin

The tests for this section can be found in the file "42sh.test.txt".

In the event of an internal error, a return code greater than 1 is returned.
Check it and check for an error message.
For example with the command:

```
$> test zzz -eq; echo $?
```

Don't be satisfied with the correction tests, do your own tests in addition. There are many possible tests for the table of hashing and the "test" builtin. Be inventive!

 Yes No

Bonus

Shell script

Test the presence and proper functioning of the shell script.

 Yes No

Parameter autocompletion

Test the presence and proper functioning of the autocompletion of the settings

 Yes No

POSIX

The entire shell must be POSIX compliant to validate this bonus.
Good luck testing POSIX compliance :)

 Yes No

Additional features

If the 42sh has additional features, post them here.
You can post up to 5 bonus features. Bonuses must be 100%. functional and do not compromise the stability of the shell.

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

Conclusion

Leave a comment on this evaluation

Finish evaluation