Assessment for vkhorkov's team

SSH repository url

git@gitlab.ucode.world:ucode/ush/vkhorkov.git

SSH

Media

ush

General

1. The goal is to share experience between the `assessor` and `defender` about the challenge.
2. Evaluate thoroughly the defending team's code in the specified repository. You are responsible for an objective grade of the team's work and knowledge.
3. Clone the repository.
4. Verify the correctness of the submitted solution according to the `Auditor` rules. If at least one rule has been violated, indicate it.
5. Files must be in the corresponding directories with names as specified in the story. If this isn't true, indicate it.
6. Be rigorous and honest, use the power of p2p and your brain.
7. If you have a disagreement, refer to the p2p and defenses documents.
8. Compile C-files with clang compiler and use these flags: `clang -std=c11 -Wall -Wextra -Werror -Wpedantic`. If the task does not compile, invalidate it.
9. Correct only the files in the cloned repository. If the story has a SUBMIT section, only the specified files should be present. If the story describes a product layout, the product must follow it. Indicate if the program does not meet at least one of these two requirements.
10. There can be several different types of questions in the protocol. Answer them according to the rules below:

- `Binary a.k.a. true-false` - mark as `True` only if everything works perfectly according to the question, or leave `False` if something fails for at least one case
- `Range 0-10` - add points strictly according to the instructions in the question
- `Label` - select one specific label according to the case detected during the entire assessment
- `Checkbox` - select all options which are appropriate to the question
- `Comment` - leave a descriptive and understandable comment to the question

11. Carry out the evaluation only in the presence of all members of the defense. Postpone assessment until all defense participants can come together.
12. Exchange knowledge during the assessment.

Assign a label below

Select the first option that came up.

Help the defending team to understand their mistakes, discuss the challenge in detail, and exchange knowledge.

Repository	The repository is empty. You have nothing to evaluate, select this item.
Auditor	There is at least one mistake according to the Auditor.
Compiling	The code in the repository does not compile.

Memory leaks	Memory leaks are present somewhere in the program. Memory has to be always freed correctly.
Crashing	The challenge solution crashed (segmentation fault, abort, bus error, uncaught exception, etc.) at some point during the correction.
Cheat	The defender used any form of cheating. Cheating involves actual, intended, or attempted deception and/or dishonest actions in relation to any academic work of ucode. Everything that is submitted must be understandable, justified, and explained.
OK	If none of the issues above were found.
Irrelevant files	The solution doesn't meet the story requirements about files or solution layout. Select this item if there are any files that are not part of the solution, or the directory layout doesn't match the one described in the story.

Act: Basic

The following set of questions tests if the project complies with the `Act` of the story.

Carry out the assessment honestly and in accordance with the challenge.

Everything must work correctly, and without runtime, compilation, or logical errors.

An honest assessment is much more valuable than overestimated or underestimated marks.

Error management

Check with command-line arguments that will produce errors.

The shell manages errors correctly like other shells do.

FALSE TRUE

Prompt

The prompt of the shell is `u$h>`.

FALSE TRUE

STDIN

The shell reads commands from the standard input:

```
>echo "ls -la" | ./ush
```

output: The `ls -la` command is properly executed in the `ush`.

Then, the prompt of the current shell is returned.

FALSE TRUE

New processes

Check the work of `u$h>` with:

#1

```
u$h> ucode
```

output: The correct error message is displayed, and then the prompt is returned.

#2

```
u$h> /bin/ls
```

output: The command `ls` is properly executed, and then the prompt is returned.

#3

```
u$h> /bin/ls -laF
```

output: The command `ls` is correctly executed with the flags `-l`, `-a`, `-F`, and, then, the prompt is returned.

Ask the defending team to show you the code where they use functions for creating new processes (`fork` + `execve` or `posix_spawn`).

Everything above works correctly and the team used functions for creating new processes.

FALSE TRUE

exit

Run the `ush` and run the following commands:

```
u$h> exit 11
```

output: The program correctly terminates and returns the parent shell.

The return value is managed correctly:

```
>echo $?
11
```

FALSE TRUE

echo

Run the `ush` again:

#1

```
u$h> echo "CBL World"
CBL World
```

output: The message is displayed correctly.

#2

```
u$h> /bin/echo "CBL World"
CBL World
```

output: The message is displayed correctly.

#3

```
u$h> echo CBL World
CBL World
```

output: (Note: write the message above without a double quotes.) It is displayed correctly.

#4

```
u$h> echo -n "\a"
```

output: There is a bell sound without a newline.

#5

```
u$h> echo "This \n must \t still \v work correctly."
...
u$h> zsh
zsh_prompt> echo "This \n must \t still \v work correctly."
...
```

output: Both messages from the defenders `echo` and `zsh echo` are the same.

#6

```
u$h> echo -E "This \n must \t still \v work correctly."
This \n must \t still \v work correctly.
```

#7

```
u$h> echo -e "rm -rf c:\\windows"
rm -rf c:\windows
u$h> echo -E "rm -rf c:\windows"
rm -rf c:\windows
```

FALSE TRUE

cd and pwd

Run the `ush` and the following commands:

#1

```
u$h> cd /specify/the/absolute/path
u$h> /bin/pwd
...
u$h> pwd
```

output: The command `/bin/pwd` confirms that the current directory has been updated.

The output from the builtin command `pwd` is the same as from `/bin/pwd`.

#2

```
u$h> cd specify/the/relative/path
u$h> /bin/pwd
```

output: The command `/bin/pwd` confirms that the current directory has been updated.

#3

```
u$h> cd
u$h> pwd
```

output: The command `pwd` confirms that the current directory is the user's home directory.

#4

```
u$h> cd -
u$h> pwd
```

output: The command `pwd` confirms that the current directory is the directory `specify/the/relative/path` used before.

#5

```
u$h> cd ~/specify/the/path
u$h> /bin/pwd
```

output: The command `/bin/pwd` confirms that the current directory has been updated.

#6

```
u$h> cd -s /tmp
```

output: There is an error that says that `/tmp` is not a directory.

The working directory is not changed.

#7

```
u$h> cd /tmp
u$h> pwd -P
/private/tmp
```

#8

```
u$h> cd -P /tmp
u$h> pwd
/private/tmp
```

#9

```
u$h> cd /var
u$h> pwd -L
/var
u$h> pwd -P
/private/var
```

Everything works perfectly.

FALSE TRUE

Escape characters

Make sure the shell allows to use escape characters.

Run the following commands in the `u$h`:

```
u$h> mkdir /tmp/dir\ \`\\name
u$h> cd /tmp/dir\ \`\\name
u$h> pwd
/tmp/dir `\name
u$h> touch \'\"\$\(\\\)\`file\ name
u$h> ls
'"$(\)`file name
u$h> cd -
u$h> rm -rf /tmp/dir\ \`\\name
```

Everything works perfectly.

FALSE TRUE

env

Run the `ush` and run the following commands:

```
u$h> env
...
```

output: Environment variables are inherited from parent shell and displayed as a `key=value` pair.

`SHLVL` is incremented by one.

Check the `env` with flags.

Start with `-i`. You can do something like:

```
u$h> env -i emacs
Set the environment variable TERM; see `tset'.
u$h> echo $?
1
```

Also try to check flag `-u`:

```
u$h> env -u TERM emacs
Set the environment variable TERM; see `tset'.
```

And the last one `-P`:

```
u$h> env -P / ls
env: ls: No such file or directory
```

Everything works perfectly.

FALSE TRUE

which

Run the `ush` and check `which` command:

```
u$h> which -s something
u$h> echo $?
1
u$h> which -s env
u$h> echo $?
0
u$h> which -a echo
echo: shell built-in command
/bin/echo
```

FALSE TRUE

export

Run the `ush` and run the following commands:

```
u$h> export ucode=cbl
u$h> env
...
ucode=cbl
...
```

output: After exporting the variable to the value, check its creation with the command `env`.

Environment variables display as `key=value`, including a new entry at the end.

FALSE TRUE

unset

Run the `ush` and run the following command (here you can use the local environment variable already created above or create a new one):

```
u$h> unset ucode
u$h> env
...
```

output: The specified variable is not in the list.

FALSE TRUE

fg

Run the `ush` and the following commands:

```
u$h> emacs filename
# press the key combination `CTRL+Z`
u$h> fg
```

The command `fg` reopens the file in emacs.

FALSE TRUE

PATH variable

Run the `ush` and the following commands:

```
u$h> date
...
u$h> unset PATH
u$h> date
...
```

output: The error message about an invalid command is displayed.

FALSE TRUE

Command separator `;`

Run the `ush` and the following commands:

#1

```
u$h> ;
```

output: The shell does nothing and returns the prompt.

#2

```
u$h> echo 1 ; pwd ; echo 2 ; ls ; echo 3 ; ls -l
```

output: These six commands are executed without any errors in the order they were written.

FALSE TRUE

Expansions

Run the `ush` and the following commands:

#1

```
u$h> ls ~
```

output: The command shows a list of files in the home directory.

#2

```
u$h> echo "The user ${USER} is on a ${SHLVL} shell level."
```

output: It displays the user login and the current shell level.

Add 5 points for each passed test.

If you find that expansion does not work perfectly in some cases, but works well in others you could add 1 point for it.

Act: Creative

The following set of questions tests creative features of the challenge.

Evaluate the quality of each feature, and whether it makes sense inside the program.

Carry out the assessment honestly and in accordance with the challenge.

Everything must work correctly, and without runtime, compilation, or logical errors.

An honest assessment is much more valuable than overestimated or underestimated marks.

Unique prompt

The shell allows to customize the prompt.

It looks unique and useful.

FALSE TRUE

Command editing

It is possible to edit written commands by moving the cursor using the keyboard.

It is user-friendly.

FALSE TRUE

Support of commands history

It is possible to find need command by using keyboard or query search with e.g. `CTRL+R`.

It is user-friendly.

FALSE TRUE

Aliases

It is possible to create an alias. Run the `ush` and the following commands:

```
u$h> mkdir test_dir
u$h> alias rd="rm -rf"
u$h> rd test_dir
u$h> ls
```

output: The folder is created, and then deleted.

FALSE TRUE

Other builtin commands

Add 1 point for each builtin command that works correctly and isn't marked above.

Shell functions

Check the following:

```
u$h> func() { echo "Goodbye Moonmen"; ls; echo "Oh goodbye"; }
u$h> func
```

output: You've just grouped three commands into one function and called it.

The output is the same as if you call these commands one by one.

FALSE TRUE

Auto-completion

Run the `ush` and run the following commands:

#1

Start command input `u$h> ec`, then press `TAB`.

output: The shell completes the command - `u$h> echo`.

#2

Start command input `u$h> z`, then press `TAB`.

output: The shell offers multiple choice to complete command.

Subsequent `TAB` keystrokes select sequentially available options.

#3

Type the following beginning of command `u$h> ema`, then press `TAB`.

output: The shell completes the command - `u$h> emacs`.

FALSE TRUE

Pipes

Run the `ush` and the following commands:

#1

```
u$h> ls /bin | cat -e
bash$
cat$
chmod$
cp$
csh$
date$
...
zsh$
```

#2

```
u$h> ls /bin | sort | uniq | head -10 | cat -e
...
```

output: The command outputs the first 10 sorted files from the `/bin` directory.

FALSE TRUE

Redirecting output

Run the `ush` and run the following commands:

#1

```
u$h> echo aaa > test
u$h> cat -e test
```

output: The file contains "aaa".

#2

```
u$h> echo Hello World > test
u$h> cat -e test
```

output: The file contains "Hello World".

#3

```
u$h> echo Bye World >> test
u$h> cat -e test
```

output: "Bye World" is appended to the file.

#4

```
u$h> cat -e < test
```

output: The command `cat` prints the contents of the file.

#5

```
u$h> cat << stop > test1
aaa
bbb
ccc
stop
u$h> cat -e test1
```

output: The file contains "aaa bbb ccc ".

Logical operators

Run the `ush` and the following commands:

#1

```
u$h> echo aaa && echo bbb
aaa
bbb
```

output: The commands work that way.

#2

```
u$h> echo aaa || echo bbb
aaa
```

output: The command works this way.

FALSE TRUE

Multiline user input

Run the `ush` and the following commands:

#1

```
u$h> "qwerty
dquote> hell yeah
dquote> "
u$h: command not found: qwerty\nhell yeah\n
```

#2

```
u$h> echo "I am so glad you've
dquote> done it!
dquote> "
I am so glad you've
done it!

u$h>
```

Everything works perfectly.

FALSE TRUE

Features

Add 1 point for each additional feature that works and isn't marked above.

CTRL+C

Run the `ush` and do the following:

#1

Press `CTRL+C`.

output: The shell just returns the prompt.

#2

Now type the random command, but press `CTRL+C` instead of running it.

output: The shell just returns the prompt.

#3

```
u$h> cat
# press key combination `CTRL+C`
...
u$h>
```

output: The shell kills the executed process and returns the prompt.

FALSE TRUE

CTRL+D

Run the `ush` and do the following:

#1

Press `CTRL+D`.

output: It exits from the current shell ([process completed]).

#2

Input data into the next file and interrupt by pressing `CTRL+D`.

```
u$h> cat
qwe
qwe
u$h>
```

output: It exits the command because of the end of the file.

FALSE TRUE

Expansions

Run the `ush` and the following commands:

#1

```
u$h> ls ~xlogin/Desktop
```

output: The command shows a list of files in the subdirectory `Desktop` of the user `xlogin` home directory.

#2

```
u$h> echo $(whoami)
```

output: It displays a login.

#3

```
u$h> echo "$(echo -n "Ave, Caesar"), $(echo -n "morituri te salutant"\!)"
Ave, Caesar, morituri te salutant!
```

#4

```
u$h> echo $PATH
```

output: The command prints the value of the `PATH` variable. Make sure the `env` is correct.

Add 2-3 points for each passed test.

If you find that expansion does not work perfectly in some cases, but works well in others you could add 1 point for it.

Reflection

Evaluate how well the defending team members understand the learning process they went through, and their progress.

Talk to the team, discuss its answers in the reflection protocol.

Evaluation

How detailed, meaningful, and clear are the responses?

Maximum grade if the answers are well-detailed, and they clearly reflect the essence of the challenge.

Minimum grade for short and/or poorly written responses.

It's okay to score low marks, the reflection process can be difficult.

Link to Reflection

Document

Check if the team has documented the challenge phases.

Documentation

The team completed documentation of the challenge.

All phases have been documented from challenge to solution.

For this, the team used the appropriate tools (README, Google Tools, Dropbox Paper, Git Wiki, Haroopad, Canva, etc.).

Code commenting is present in the implementation.

So, how do you like the documentation for this project?

Share

Check if the team has shared the information about this challenge.

Publishing

The team shared its solution with the world.

They have shared their work on GitHub/GitLab/BitBucket or something similar.

They wrote an article, or a post on social media about the challenge.

If at least one of these two statements is true, mark as true.

FALSE TRUE

Feedback

Your feedback on the evaluation.

Comment

Leave a comment on this evaluation.

Comments *

Nice

Finish Assessment