Assessment for vbalachevs's group

https://:@gitlab.ucode.world:8443/unit-factory-kyiv/ush/vbalachevs-2.(

Media

ush

General

1. The goal is to share the experience with `assessor` and `defenders` about this challenge.
2. You have to check really carefully defending team's code in the specified repository. Team's knowledge depends on your evaluation.
3. Clone the repository.
4. You must verify the correctness of the challenge according to the `Auditor` rules. If at least one rule is violated, you must indicate this.
5. Files must be in the corresponding directory with names as specified in the story, if it is not true you must indicate it.
6. Be rigorous and honest, use the power of p2p and your brain.
7. Read the docs about p2p and defenses, if you have any disputes.
8. You must use the following flags to compile: `clang -std=c11 -Wall -Wextra -Werror -Wpedantic`. If the challenge does not compile you must indicate it.
9. You must check available work on the repository. Only files from SUBMIT part of the story must be assessed. You must indicate it if there are files that wasn't mentioned in the SUBMIT part.
10. There are 4 different type of questions in the protocol. You must check them according to the rules below:
    - `Binary a.k.a. true-false` - you must mark as `True`, only if everything works OK according to the questions, but if something fails at least one case, you must mark as `False`;
    - `Range 0-10` - you must add points strictly according to the instructions in the question;
    - `Label` - you must select a specific label according to the case detected during the whole assessment;

- `Comment` - leave a descriptive and understandable comment according to the question.

11. Evaluation is carried out only in the presence of all members of the defense. Postpone assessment until all defense participants can come together.
12. Gain the knowledge and share your knowledge during the assessment.

Assign label below

Choose the first option that came up.

Help defending team to understand mistakes, discuss challenge in details, exchange of your knowledge.

REPOSITORY	Clone team repository. Is repo empty? If the repository is empty, you have nothing to evaluate, you should choose this item.
AUDITOR	Check available files due to Auditor. In case of any errors, you should choose this item.
COMPILING	Does the `Makefile` match the rules of the Auditor? Makefile should contain `all`, `install`, `uninstall`, `clean` and `reinstall` targets. Is there `-std=c11 -Wall -Wextra -Werror -Wpedantic` for compilation?
MEMORY LEAKS	Memory leaks are present somewhere in the program. Memory has to be always freed correctly.
CRASHING	There is a crash (segmentation fault, abort, bus error, etc.) somewhere in the program.
EXTRA FILES	Check available work on the repository. You must indicate it if there are files that wasn't mentioned in the SUBMIT part.
CHEAT	Cheating involves actual, intended, or attempted deception and/or dishonest action in relation to any academic work of the ucode. Everything that is submitted must be understandable, justified and explained.
OK	If none of the items above are suitable.

Act: Basic

This is always what is clearly stated in the story. The assessment should be set honestly in accordance with the challenge.

Everything should work correctly, without runtime, compilation or logical errors.

The real assessment is much more valuable than overestimated or underestimated marks.

Error management

A very important part of software development is error management.

You always should consider during development that program/function can receive invalid input data.

Check for command-line arguments that will produce errors.

The shell must correctly manage errors as other shells do.

FALSE TRUE

Prompt

Check the prompt of the shell.

Does it look like `u$h>`?

FALSE TRUE

fork and execve

Check whether `u$h>` works correctly with:

#1

```
u$h> ucode
```

output: The correct error message must be displayed, and then return the prompt.

#2

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

output: The command `ls` must be properly executed and then return the prompt.

#3

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

output: The command `ls` must be correctly executed with the such flags as `-l`, `-a`, `-F` and then return the prompt.

Ask the defending team to show you code, where they use fork and execve.

Does everything from above work correctly and a team used fork and execve?

FALSE TRUE

exit

Run the `ush` and run the following command:

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

output: The program must exit correctly and return the parent's shell.

Return value must be managed correctly:

```
> echo $?
11
```

FALSE TRUE

echo

Run the `ush` again:

#1

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

output: The message must be displayed correctly.

#2

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

output: The message must be displayed correctly.

#3

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

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

#4

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

output: There must be a bell sound without 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 corre
...
```

output: Both messages from the defenders `echo` and `zsh echo` must be 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 run the following commands:

#1

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

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

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

#2

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

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

#3

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

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

#4

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

output: The command `pwd` must confirm 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` must confirm that the current directory has been updated.

#6

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

output: There must be an error which tells that `/tmp` is not a directory.

Working directory must not be 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
```

Set this as `OK` only if everything is perfect.

FALSE TRUE

Escape characters

Check whether shell allows usage of escape characters.

Run the following commands in the u$h:

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

Mark this as `True` only if everything is perfect.

FALSE TRUE

env

Run the `ush` and run the following command:

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

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

`SHLVL` must be incremented by one.

Now please check `env` with flags.

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

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

Also try to check flag `-u`:

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

And the last one `-P`:

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

Set this as `OK` only if everything is perfect.

FALSE TRUE

which

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

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

FALSE TRUE

export

Run the `ush` and run the following command:

```
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 should 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 must not be in the list.

FALSE TRUE

fg

Run the `ush` and run the following commands:

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

output: The command `fg` must reopen the file in emacs.

FALSE TRUE

PATH variable

Run the `ush` and run the following command:

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

output: the error message about invalid command must be displayed.

FALSE TRUE

CTRL+C

Run the `ush` and run the following command:

#1

Press `CTRL+C` instead of typing the command.

output: The shell must just return the prompt.

#2

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

output: The shell must just return the prompt.

#3

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

output: The shell must kill executed process and return the prompt.

FALSE TRUE

CTRL+D

Run the `ush` and run the following command:

#1

Press `CTRL+D` instead of typing the command.

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

#2

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

```
u$h> cat > test2
qwe
```

```
qwe
u$h>
```

output: It must exit the command because of the end of the file.

FALSE TRUE

Command separator `;`

Run the `ush` and run the following command:

#1

```
u$h> ;
```

output: The shell must do nothing and return the prompt.

#2

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

output: These 6 commands must be executed without any errors in the order they were written.

FALSE TRUE

Expansions

Run the `ush` and run the following command:

#1

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

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

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

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

#2

```
u$h> echo `whoami`
```

output: It must display login.

#3

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

#4

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

output: It must display the user login and the current shell level.

#5

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

output: The command must print value of `PATH` variable. Check `env` whether it is correct.

Add 2 points for each passed test.

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

Act: Creative

This is always what your brain will come up with.

The assessment must be honest in accordance with the quality and appropriateness of the assignment.

Everything should work correctly, without runtime, compilation or logical errors.

A real estimate is much more valuable than overestimated or underestimated estimates.

Feature just for the feature is not a product-oriented approach.

Unique prompt

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 keyboard.

Is it 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`.

Is it user-friendly?

FALSE TRUE

Aliases

It is possible to create alias. Run the `ush` and run the following command:

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

output: Make sure that the folder was created, and then deleted.

FALSE TRUE

Other builtin commands

Were any builtin commands implemented that were not marked above?

Add 1 points for each commands that works correctly.

Shell functions

Can you group commands into functions?

Please, go through this test:

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

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

Output must be the same as if you call these commands one by one.

FALSE TRUE

Auto-completion

Run the `ush` and run the following command:

#1

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

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

#2

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

output: The shell must offer multiple choice to complete command.

Subsequent `TAB` keystrokes must select sequentially available choices.

#3

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

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

FALSE TRUE

Support of pipes

Run the `ush` and run the following command:

#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: Command must output first 10 sorted files from `/bin` directory.

FALSE TRUE

Redirecting output

Run the `ush` and run the following command:

#1

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

output: The file must contain "aaa".

And run the next:

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

output: The file must contain "Hello World".

#2

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

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

#3

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

output: The command `cat` must print the contents of the file.

#4

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

output: The file must contain "aaa bbb ccc ".

Logical operators

Run the `ush` and run the following command:

#1

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

output: Make sure the teams work that way.

#2

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

output: Make sure the command works this way.

FALSE TRUE

Multiline user input

Run the `ush` and run the following command:

#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>
```

Mark this as complete only if everything is perfect.

FALSE TRUE

Features

Were any additional feature implemented that were not marked above?

Make sure they are really useful and make sense.

Feature just for the feature it is not product-oriented approach.

Add 1 points for each one that works correctly.

Share

Have the team shared the solution with the world?

Check if they posted solution on GitHub/GitLab/BitBucket, etc. Perhaps they wrote a post in any of the social networks where they talked about the challenge.

Have they written an article? If at least one of these statements is true, then rate it.

FALSE TRUE

Reflection

This part is very important! You must evaluate how truly team that you assess understand the process of learning.

Does the team rightly evaluate its progress? Do students understand what they have learned during the challenge?

Talk with the defending team, discuss answers in the reflection protocol.

Evaluation

Use the slider to rate the responses.

Max rate if answers are detailed and they clearly reveal the correct essence, which corresponds to the theme of the task, Min rate - otherwise.

Be careful, low marks it's ok, study isn't always easy!

LINK TO REFLECTION

Feedback

Your feedback about the evaluation is very important

Comment

Leave a comment on this evaluation

Comments*

0 / 140

FINISH ASSESSMENT