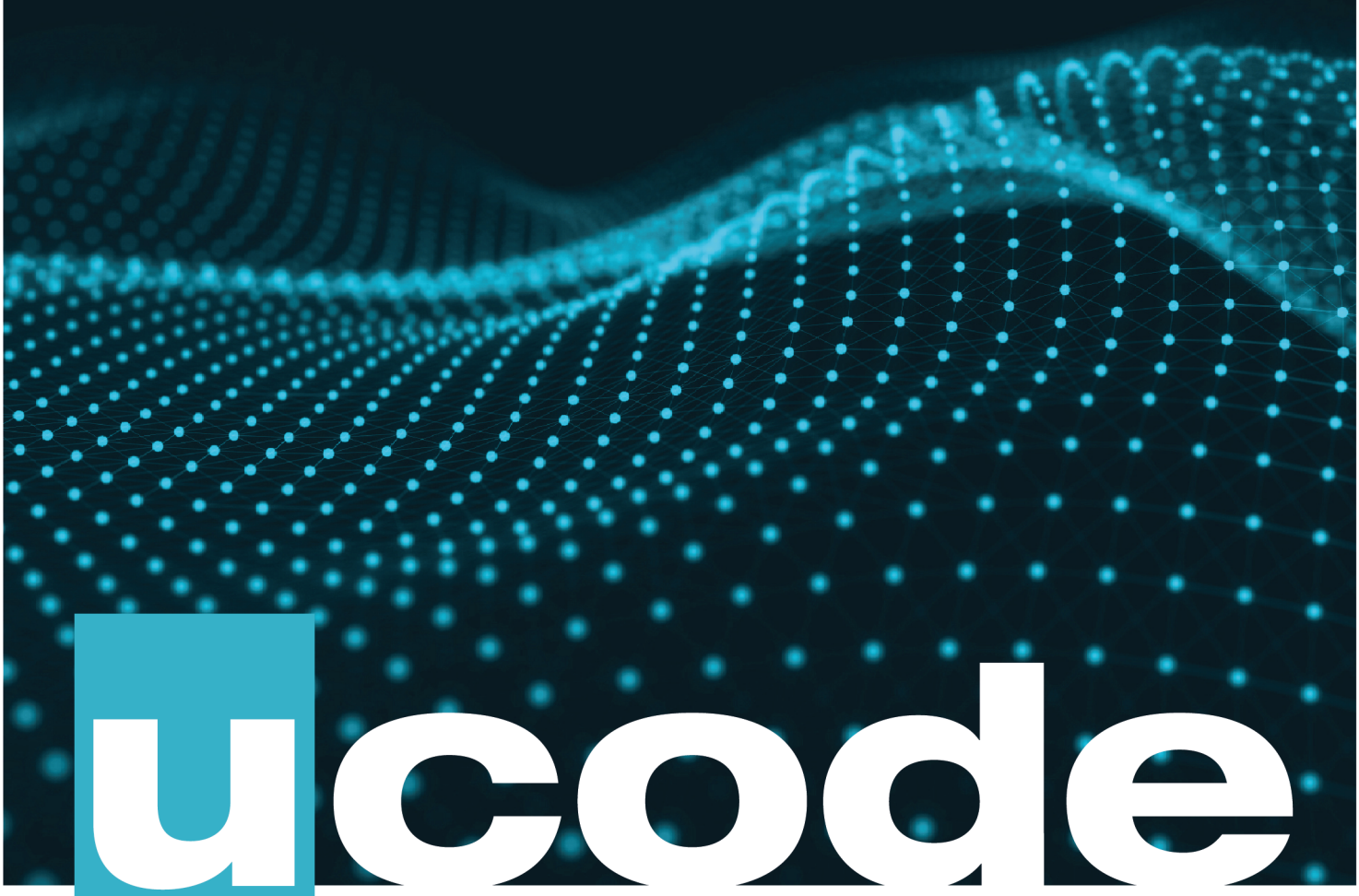
ush

Track C

February 10, 2020



ucode

Contents

ucode

Engage

DESCRIPTION

Have you ever thought what is the user interface (UI) in a wide sense?
Did you know that it's not just about computer and software design?
There are many physical interfaces that surround us in everyday life, such as water taps, handles, keyboards, etc. Basically, everything can be called as an interface if it helps the user to interact with the object/device. However, the term UI is often used in the sense of software and electronic devices and provides a layer between human and computer. There are command-line interfaces (CLI), graphical user interfaces (GUI), text user interfaces (TUI) and much more can be found here that can be used during software design.

Terminal is probably one of the very first tools you've started to use during the education.
Do you prefer `Terminal`, `iTerm` or `cmd.exe` to execute shell?
What is your favorite shell `zsh`, `bash`, `csh` or `powershell`?
A Unix shell is a command-line interpreter that provides a command line user interface for Unix-like operating systems. The most generic sense of the term shell means any program that users employ to type commands. A shell hides the details of the underlying operating system and manages the technical details of the operating system kernel interface, which is almost the lowest-level component of most operating systems.

The previous projects have already given you basic understanding of a Unix systems.
In this challenge, we will go somewhat further. Challenge invites you to find out how `shell` works internally. You will deepen understanding of OS architecture and application programming interface. Also you've got an opportunity to develop your ideal shell(or not).

Imagine you are a computer science pioneer in 1960s. GUI has not been invented yet. You need to develop user-friendly interface to execute commands to the computer.

BIG IDEA

User interface.

ESSENTIAL QUESTION

How users can interact with OS kernel?

CHALLENGE

Develop Unix shell.

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. This will help you and your team realize what knowledge you will get from this challenge and how to move forward.

Ask your neighbor on the right, left, or behind you, and discuss the following questions together. You can find the answers in the Internet and share it with student around you.

We encourage you to find answers as many questions about `shells` and `Unix` as possible. Note down your discussion.

- What is the term process in computing mean?
- How does shell manage commands?
- What is the step-by-step algorithm of the shell (reading, interpreting, and execution of the commands)?
- What is shell builtins? Why does shell require them?
- How to run external program in separate process?
- What are the pros and cons of builtins comparing to external programs?
- How to create new process?
- What is the daemon processes?
- What shell features are needed first of all?
- What does prompt stand for?
- What is the difference between terminal, shell and console?
- What are the environment variables and how shell interacts with them?
- What is POSIX?
- What does C standard library for POSIX systems consist of?

GUIDING ACTIVITIES

These are only a set of example activities and resources. The goal is to develop a solution. Do not forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

1. Meet the team.
2. Read the story to the end and carefully study this task.
3. Read the docs of shells (e.g. sh, zsh, csh, bash) and try to understand the differences.
4. Experiment with standard shells. Learn their features. There you'll find out additional information that is not listed in the docs.
5. Read about exotic shells.
6. Use existing shell (e.g. zsh) as a reference for this challenge.
7. Read the documentation about every component of the solution.

8. Collaborate with other students to investigate challenge in-depth. Learn from others and share your knowledge.
9. Build a great team. A great team is the one whose work is based on clear objectives, clear roles, clear communication, cooperation, and opportunities for personal development.
10. Cooperate with each other. Teamwork is not about individual achievements, it's about what the group accomplishes together.
11. Establish communication. Get to know each other closer.
12. It is important to set clear milestones and deadlines for the process and products. We strongly recommended using a task manager to organize processes in the team (e.g. Trello, Jira, etc). Use meetings for periodic monitoring of the deadlines and for the development process.
13. Clone your git repository, that is issued on the challenge page.
14. Try to choose the best solutions.
15. Distribute tasks between all team members.
16. Start to develop the solution. Offer improvements. Test your code.
17. Explore new things for you. Talk, discuss and communicate.
18. Start developing your implementation.
19. Have fun :)

Building an effective team is one of the most important responsibilities for all of us. It's not something you can instantly achieve. It's an ongoing process, which requires constant attention and evaluation.

ANALYSIS

You need to analyze all the collected information before you start.

- Be attentive to all statements of the story.
- Perform only those tasks that are given in the story.
- You can proceed to `Act: Creative` only after you have completed all requirements in `Act: Basic`.
- The challenge must be performed in `C`.
- Your challenge must have the next structure:
  - `src` directory contains source files `.c`;
  - `obj` directory contains object files `.o`. It must be created only during compilation;
  - `inc` directory contains header files `.h`;
  - `libmx` directory contains source files of your library including its `Makefile`. It is recommended but not mandatory;
  - `Makefile` that compiles the library `libmx` firstly (if it exist) and then compiles and builds `ush`.

- Makefile must be written in accordance to Auditor.
- You should submit only files required to complete the task in the required directories and nothing else. Garbage shall not pass.
- You should compile C files with clang compiler and use these flags: `-std=c11 -Wall -Wextra -Werror -Wpedantic`.
- Your program must manage memory allocations correctly. Memory which is no longer needed must be released otherwise the task is considered as incomplete.
- Usage of forbidden functions is considered as cheat and your challenge will be failed.
- You must complete tasks according to the rules specified in the `Auditor`.
- Your challenge will be checked and graded by students. The same as you. `Peer-to-Peer (P2P) learning`.
- Also, your challenge will pass automatic evaluation which is called `Oracle`.
- Got a question or you do not understand something? Ask the students or just Google that.
- Use your brain and follow the white rabbit to prove that you are the Chosen one!!!

Act: Basic

ALLOWED FUNCTIONS

C standard library

BINARY

ush

DESCRIPTION

Develop basic command-line interpreter. Before you start to implement exotic shell of your choice you need to add must-have features without which no shell can exist.

1. The default prompt must look like `u$h>` followed by the space character.
2. The shell must deal only with one line user input. In other cases, appropriate descriptive error message must be displayed.
3. The shell must implement builtin commands without flags: `export, unset, fg, exit`.
4. The shell must also implement the following builtin commands with flags:
   - `env` with `-i`, `-P`, `-u`;
   - `cd` with `-s`, `-P` and `-` argument;
   - `pwd` with `-L`, `-P`;
   - `which` with `-a`, `-s`;
   - `echo` with `-n`, `-e`, `-E`;
5. The shell must call the builtin command instead of binary program if there is the name match between them.
6. The shell must correctly manage errors like other shells do.
7. The shell must manage user environment correctly.
8. The shell must run programs located in the directories listed in the `PATH` variable.
9. The shell must manage signals `CTRL+D`, `CTRL+C` and `CTRL+Z`.
10. The shell must implement command separator `;`.
11. These characters must be escaped to be used literally: `space`, `'`, `"`, `$`, `(`, `)`, `\`, `` ` ``, `{`, `}`.
12. The shell must manage these expansions correctly:
    - tilde expansion `~` with the following tilde-prefixes: `~`, `~/dir_name`, `~username/dir_name`, `~+/dir_name`, `~-/dir_name`;
    - the basic form of parameter expansion `${parameter}`;
    - command substitution `` `command` `` and `$(command)`.

The purpose of this challenge is to study the system's API, thus you are allowed to use any C standard library functions. This implies that you are forbidden to use any third-party libraries except it is clearly stated in the story. If you need some, you have to develop it and consider adding it into the libmx for future usage.

CONSOLE OUTPUT

```
>./ush
u$h> unknown
ush: command not found: unknown
u$h> echo $SHLVL
2
u$h> cd ~xlogin
u$h> pwd
/Users/xlogin
u$h> cd /usr/local/bin
u$h> pwd
/usr/local/bin
u$h> echo "${PWD}"
/usr/local/bin
u$h> mkdir tricky\ dir
u$h> touch tricky\ dir/more\ tricky\`file
u$h> ls tricky\ dir/
more tricky`file
u$h> rm -rf tricky\ dir
u$h> echo "Effective user id: `whoami`"
Effective user id: xlogin
u$h> cd -
u$h> rm -rf ush
u$h> echo "Bye!
Odd number of quotes.
u$h> echo 'F*ck. Bye!'
F*ck. Bye!
u$h> exit
>
```

FOLLOW THE WHITE RABBIT

- man zsh
- man builtin
- man zshbuiltins
- man env
- man environ
- man fork
- man execve
- man 2 wait
- man 2 stat
- man 3 exit
- man 3 signal
- man 2 kill
- man getcwd

- man 2 chdir
- man access
- man opendir

Act: Creative

ALLOWED FUNCTIONS

C standard library, termcap library

DESCRIPTION

You can proceed to this part only after you have completed all requirements above.

Note that `Act: Basic` gives the minimum points to validate the challenge. You need to implement at least a few items from this part to succeed the challenge. That is the place where your imagination and creativity plays a major role.

Below you can find an example list of features which could be added to your shell. You can implement any features you like but make sure they are really useful and make sense. Explore something meaningful in already existing shells or perhaps you can invent your functionality. Feature just for the feature it is not product-oriented approach.

Examples grouped by complexity to orient you.

- Allow users to customize prompt and to make it unique and useful e.g. show current directory, git info, etc.
- Implement command editing. Move cursor using `Arrow` keys or `HOME` and `END` keys.
- Add support of commands history. Sequential search e.g. with `Page Up` and `Page Down` keys. Or query search with `CTRL+R`.
- Implement more builtins of your choice.
- Implement multiline user input.
- Add support of shell functions. `tripple_ls() { ls; ls; ls; }`
- Implement auto-completion using `TAB` key.
- Add support of pipes `|`.
- Add support of redirecting output `>, <, >>, <<`.
- Add support of logical operators `&&` and `||`.
- Add support of aliases.
- Any other useful features you like.

And remember, you do not need to create a full-featured shell. Orient on the approximate time of challenge accomplishment, on the challenge web page.

SEE ALSO

`man termcap`

Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences.

During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- A text post, as a summary of your reflection;
- Charts, infographics or other ways to visualize your information;
- A video, either of your work, or a reflection video;
- An audio podcast. Record a story about your experience;
- A photo report with a small post.

Helpful tools:

- Canva - a good way to visualize your data.
- QuickTime - an easy way to capture your screen, record video or audio.

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends;
- YouTube - upload an exciting video;
- GitHub - share and describe your solution;
- Telegraph - create a post that you can easily share on Telegram;
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.