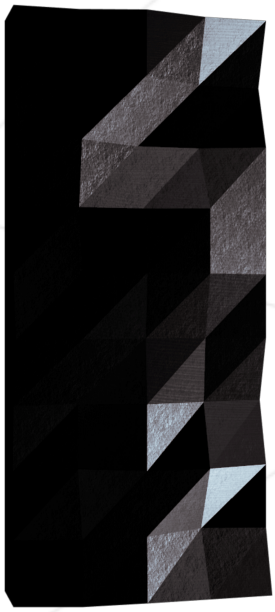


UNIX leaning project

ft_select

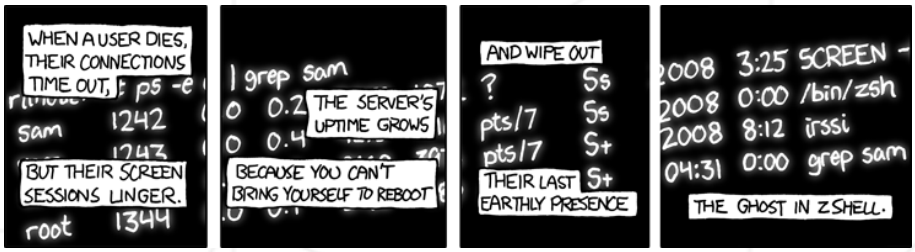
Pedago team pedago@42.fr

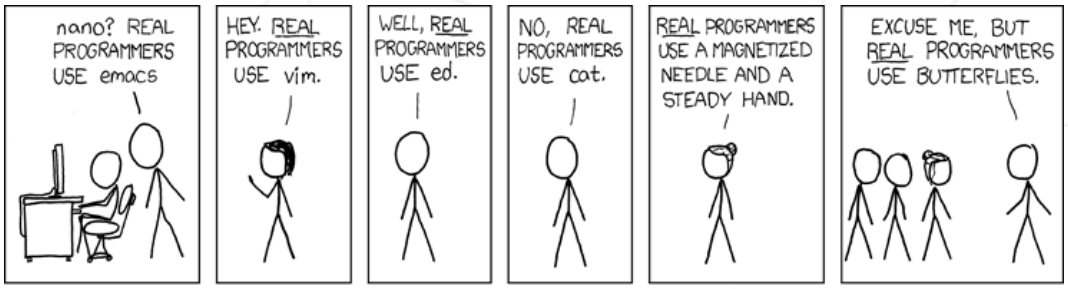
Summary: This project is meant to make you code a small program using termcaps which will allow you to pick from a list of choices and return it to your shell.

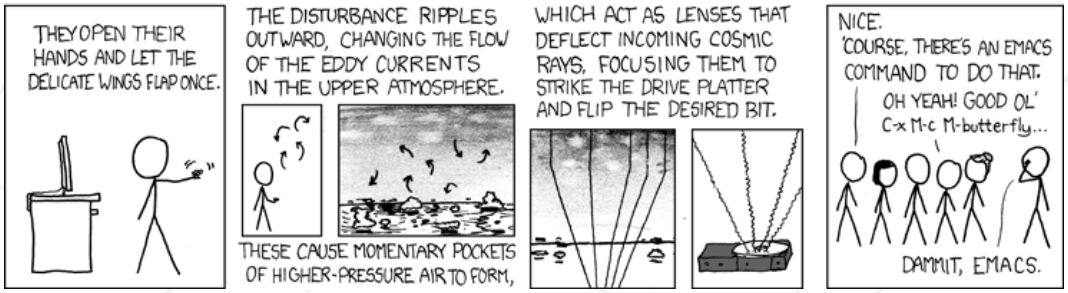
Contents

Chapter I

Foreword







Chapter II

Introduction

Some programs, such as your shell, `aptitude`, `top`, `tig`, `mcabber`, `dwarf fortress`, `zangband` or `herrie` have at least two things in common: Firstly these programs are all executed in a terminal and secondly they all offer an advanced user interface despite the fact that they don't offer the `Microsoft Windows` and `OSX`'s windowed graphic interface that you have come to expect.

Creating a user interface for a program that is executed in a terminal is doable. It requires that you master the programming work involved as a terminal does not do much in its "rough" mode. To see for yourself, launch the command cat without any arguments and press any keys or a combination of keys on your keyboard...

In doing this, as long as you press alphanumeric keys, nothing special happens. However, if you press either the arrows, the `esc` key or the `fn` keys, for example, some random characters will appear... redo the same thing in your shell now. You are used to the left arrow moving the cursor to the previous column. Why do we observe such differences between `cat` and your shell?

Your terminal manages many things for you without you noticing, such as for instance the display of characters as you type them or the bufferisation of each line.

If, by now, you are asking yourself how you can control your terminal, this project is made for you. You will learn to configure your terminal via the "`struct termios`" structure as well as to use its technical capabilities, the distinguished "`termcaps`".

Chapter III

Technical considerations

- You cannot use any global variables, except for those which have already been defined for you. An exception will be made when managing the signals, even though there are other ways to do it.

- A local function to a `C` file (which therefore isn't prototyped in the file .h) must be defined as `static` in accordance to the Norm.

- You need to pay attention to types and use casts wisely when deemed necessary, in particular when a void* type is involved. In absolute terms, avoid implicit casts, no matter which types.

Chapter IV

General Instructions

- This project will be corrected by humans only. You're allowed to organise and name your files as you see fit, but you must follow the following rules.
- The executable file must be named `ft_select`.
- Your `Makefile` must compile the project and must contain the usual rules. It must recompile and re-link the program only if necessary.
- If you are clever, you will use your library for your `ft_select`. Submit also your folder `libft` including its own `Makefile` at the root of your repository. Your `Makefile` will have to compile the library, and then compile your project.
- Your project must be written in accordance with the Norm. Only norminette is authoritative.
- You have to handle errors carefully. In no way can your program quit in an unexpected manner (Segmentation fault, bus error, double free, etc).
- Your program cannot have memory leaks.
- You'll have to submit a file called `author` containing your username followed by a '\n' at the root of your repository.

```
$>cat -e author
xlogin$
```

- A project that displays abstract art (strange characters and/or inadequate) is considered non-functioning.
- In case your program is killed while executing, your terminal must naturally operates normally afterwards...
- You are Never allowed to submit a code that you did not write yourself. If any doubt exist, you will be invited to recode it to defend your good faith.
- The `termcap` library is mandatory for this project.

- Before you ask, the `ncurse` library is forbidden for this project. Else there would be no challenge...
- This project demands a good thought-process. It is absolutely vital that you discuss it amongst yourselves and that you share your info!
- Within the mandatory part, you are allowed to use only the following libc functions:
  - isatty
  - ttyname
  - ttyslot
  - ioctl
  - getenv
  - tcsetattr
  - tcgetattr
  - tgetent
  - tgetflag
  - tgetnum
  - tgetstr
  - tgoto
  - tputs
  - open
  - close
  - write
  - malloc
  - free
  - read
  - exit
  - signal
- You can ask your questions on the forum, on slack...

Chapter V

Mandatory part

- Write an “ft_select” program that will get a series of arguments. The list of arguments is displayed in your terminal.
- The user can move through the list of arguments using arrows (the list is circular).
- One or more choices can be selected or un-selected with the the `space` key. With each selection, the cursor must automatically position itself on the next element.
- When the user validates the selection with the `return` key, the list of choices will be sent back to shell. The latter (i.e. list of choices sent back) will be separated by a space. This will allow you to use your program ft_select inside a shell script (to create a “set” for example).
- We must be able to write the following commands:

```
$> set reponse = `ft_select choice1 choice2 choice3 choice4`

$> more `ft_select *.c`

$> rm `ft_select ~/*`   NB: ONLY use this command IF you
                        are CERTAIN that your ft\_select
                        works. We deny any responsibility
                        if your your ft\_select is
                        faulty...
```

- You will also need to manage the resizing of the window by the user. The list must be seen on a few columns if the size of the windows does not have enough lines to show them all in one column. If the user resizes the window while he is using it, the ensemble of his choices will have to reposition automatically and the selected choices will have to remain. The cursor of selection must be repositioned in a reasonable manner after a resizing.
- If the window is too small to show everything (not enough lines and/or columns), the program must react reasonably as long as the dimensions of the window aren’t sufficient. Under no circumstances, can the program quit or crash. As soon as the window is large enough, the program must function properly again.

- If the user presses `esc` key, the program musn't send anything to the shell and it has to exit properly.
- If the user presses either `delete` or `backspace`, the element the cursor is pointing to must be erased from the list. If there are no more elements in the list, the behavior must be exactly the same as if the user had pressed `esc`.
- Choice non selected: normal text.
- Selected choice: inverse video text.
- Cursor's position: underlined text.
- Selected choice + cursor's position: inverse video underlined text.
- Whichever way your program ends, the default configuration of your terminal **MUST** be restored. This is true even after it received a signal (except for the signals that we cannot intercept, but this would mean that your program does not work).
- We must be able to interrupt your program with `ctrl+z` and restore it with `fg` without seeing any changes in its behavior.
- If the program is launched in an empty environment, you need to behave reasonably.

Chapter VI

Bonus part

We will look at your bonuses if and only if your mandatory part is EXCELLENT. This means that your must complete the mandatory part, beginning to end, and your error management must be flawless, even in cases of twisted or bad usage. If that's not the case, your bonuses will be totally IGNORED.

Possible bonuses :

- When the window is too small, the columns move from left to right depending on the position of the cursor.
- When the program is terminated, what needs to be erased must be, and the prompt as well as the cursor will appear on the line that follows the program's call. Run `tig` to see what I mean. Don't forget about the signals!
- A beautiful interface (up to the corrector to decide, not you!)
- If the choices are files names, colorize the list according to the extensions (a bit like `ls -G` on `OSX`).
- Positioning of the cursor when we type a sequence of characters that matches an element in the list (dynamic search).

Chapter VII

Submission and peer correction

Submit your work on your `GiT` repository as usual. Only the work on your repository will be graded.

Good luck to all and don't forget your author file!