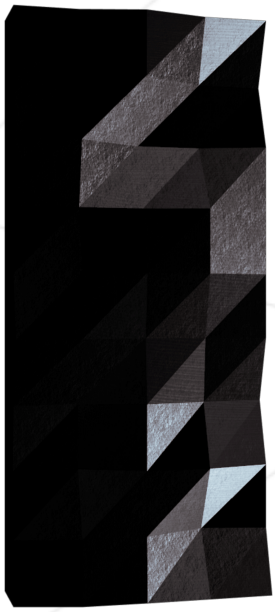

21sh

The vengence's return

Staff pedago pedago@42.fr

Summary: You'll have to start from your minishell and make it stronger to get little by littler closer to a real functionnal shell. You'll add couple of features such as multi-commande management, redirections as well as line edition that will allow you to use arrows for example.

Contents

Chapter I

Foreword

Insert foreword here

Chapter II

Introduction

Thanks to the `Minishell` project, you discovered a part of what is behind the scene of a shell, such as the one you use everyday. And more specifically the process' synchronisation creation with functions like `fork` and `wait`.

The `21sh` project will make you go further by adding, amoungst other things, inter-process communication using pipes.

You'll discover, or rediscover if you worked on the `ft_select` project, `termcaps`. This library will allow you to add to your shell a line edition feature. You'll then be able to edit a typo made on your command without having to retype it completely as well as repeat a previous command using history. Of course you'll have to code those features. Good news is `termcaps` will help you do it, bad news is it'll not do it for you!

Chapter III

Objectives

`Unix` programmation is great, The school's 3 shell projects allow you to discover a big part of the system's API and it can only be good for you.

However, the shell projects are commands interpreter above all and initiate you to a very important part of IT: compilation. Interpreter are programs that read and execute other programs, unline compilators which translate other programs into other langages. Interpreters and compilators have more in common than they have differences though: whether it comes to executing or translating a program, first we need to understand the program itself, and be able to detect and reject malformed programs.

You probably know this already, but the set of commands we can send to a shell form a langage. This langage has lexical, syntactic and semantical rules, that will have to be respected by your 21sh by going through a set of very precise steps well documented on the internet. For example the "2.10 Shell Grammar" section of this document.

The key to a successful `21sh`, and `42sh` later on is a clear and well managed code organisation. Be sure that a simple space based `split` on your command line will not do the trick here. To avoid losing time, consider this solution as a one way ticket to disaster.

Here are couple of key words that i suggest you to properly understand: "lexical analysis", "lexer", "syntactic analysis", "parser", "semantic analysis","interpreter", and of course "abstract syntax tree" (or "AST").

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 `21sh`.
- 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 `RTv1`. 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.

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

- Within the mandatory part, you are allowed to use only the following libc functions:
  - `malloc`, `free`
  - `access`
  - `open`, `close`, `read`, `write`
  - `opendir`, `readdir`, `closedir`
  - `getcwd`, `chdir`
  - `stat`, `lstat`, `fstat`
  - `fork`, `execve`
  - `wait`, `waitpid`, `wait3`, `wait4`
  - `signal`, `kill`
  - `exit`
  - `pipe`
  - `dup`, `dup2`
  - `isatty`, `ttyname`, `ttyslot`
  - `ioctl`
  - `getenv`
  - `tcsetattr`, `tcgetattr`
  - `tgetent`
  - `tgetflag`
  - `tgetnum`
  - `tgetstr`
  - `tgoto`
  - `tputs`
- You are allowed to use other functions or other librairies to complete the bonus part as long as their use is justified during your defense. Be smart!
- You can ask your questions on the forum, on slack...

Chapter V

Mandatory part

To begin with, every `minishell` features are implicitly part of the `21sh` mandatory part. Furthermore you'll have to add the following new features:

- A line edition feature using the `termcaps` library. Check the following description below.
- The `ctrl+D` et `ctrl+C` keys combination features in line edition and process execution.
- The "`;`" command line separator
- Pipes "`|`"
- The 4 following redirections "`<`", "`>`", "`<<`" et "`>>`"
- File descriptor aggregation, for example to close the standard error output:

```
$> ls
riri
$> rm riri; cat riri 2>&-
```

Here is a representativ example of commands your `21sh` must be able to execute correctly:

```
$> mkdir test ; cd test ; ls -a ; ls | cat | wc -c > fifi ; cat fifi
.   ..
       5
$>
```

Regarding the line edition, you must at least manage the followin features. The keys to be used are used as examples, you're free to use other ones as long as your shell remains logical and intuitive. The corrector will decide what's logical and intuitive, so be careful not to get carried away with creativity.

- Edit the line where the cursor is located.

- Move the cursor left and right to be able to edit the line at a specific location. Obviously new characters have to be inserted between the existing ones similarly to a classic shell.

- Use up and down arrows to navigate through the command history which we will then be able to edit if we feel like it (the line, not the history)

- Cut, copy, and/or paste all or part of a line using the key sequence you prefer.

- Move directly by word towards the left or the right using `ctrl+LEFT` and `ctrl+RIGHT` or any other reasonable combination of keys.

- Go directly to the beginning or the end of a line by pressing `home` and `end`.

- Write AND edit a command over a few lines. In that case, we would love that `ctrl+UP` and `ctrl+DOWN` allow to go from one line to another in the command while remaining in the same column or otherwise the most appropriate column.

- Completely manage quotes and double quotes, even on several lines (expansions excluded).

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.

There are quite a few features that will appear in 42sh. Here is however a list of bonuses that you can implement immediately:

- Search through history using `ctrl+R`
- Implement a hash table for binary files
- Simple or advanced completion using `tab`.
- `Emacs` and/or `Vim` binding mode freely activable or deactivable.
- Syntactic shell coloration freely activable or deactivable.
- Any additional bonus that you will feel useful.

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!