Unix system

42sh

Contact b-psu-050@epitech.eu

Abstract:

Contents

.1 Administrative details

- The sources must be turned-in on the `PSU_year_42sh` directory.
  ex: PSU_2013_42sh for the 2013-2014 scolar year
- The binary's name is not important.
- Your binary shall be compiled using a Makefile.
- Questions can be asked in the forum, in the "42sh" thread of the System Unix B2 module.
- The teachers' answers are the only ones that you must consider as official.
- You are advised to work with other groups and to compare your results. However, each group must have its own implementation.
- You must have an author file with one login (and only one) by line
- Each member shall be able to explain the global functioning and the architecture of the project (the used structures and the meaning of each part). You must also be able to explain what you did yourself and modify it or make it again during the oral defense.
- You can exclude people from your team until 2 months before the turn-in date. We will not pay attention to any request after this deadline and each group member must agree to make this decision.
- You don't need to have a group leader but we highly recommend it.
- Everything you need must be located into the turn-in repository and its subdirectories (nothing elsewhere, even with full rights and absolute paths)

.2 Subject

- The goal here is to write a Unix Shell.
- The project is composed with 2 parts detailed below:
  - A **mandatory** part, which must be done imperatively and which worth 8 points
  - An **optional** part, which will be taken into account only if the mandatory one is full and functional.
- The stability and usability of the whole of your project will be highly taken into account. It is advised to conform to the customs and habits.

.3 Mandatory part

- This part must be **FULLY FUNCTIONNAL**. Otherwise, your score will be 0.
- A minimum line reader which:
  - displays a prompt (more or less sophisticated)
  - retrieves the line entered (a simple get_next_line(0) can be enough)
- Commands execution with its parameters

```
ex: \$ls -l /
```

  - Correct management of spaces and tabulations
  - Correct management of the PATH (a cache system is not mandatory)
  - Correct handling of errors and return values

```
ex: $./str_maxlenoxc "ddd" "dd" "who"
     segmentation fault (core dumped)
    $
```

- Redirections:

```
ex: \$</etc/hosts od -c | grep xx | wc >> /tmp/z -l
```

  - « < > »
- Pipes

- Builtins:
  - cd (with a single cd and cd -)
  - echo
  - exit
- Separators :
  - ;
  - &&
  - ||

.4 Optional part

It is on this part that you should earn the majority of the points. It is globally free. You can do whatever you want. However, the consistency of the whole will be taken into account.
Again, the stability is much more important than the quantity. Do not include an option that may trouble the rest of the program (especially the mandatory part). First of all, think about the usability and the stability.

List of advised options:

- Inhibitors " ’

  ex: `\$ls "who|’"  ’"’"slt\""`

- Globing * ? [ ]

  ex: `$echo {a*[^c],b??.*[a-z]}/b*.{c,h}`

- Background process

  ex: `$sleep 100 &`

- The ‘ (magic quotes)

  ex: `$kill -9 ‘ps ax | grep netscape | awk ’{print $1}’‘`

- Parenthesis: ()

  ex: `$(cut -d\  -f2 .note | tr ’\n’ +;echo 0)| bc -l`

- Variables (local and env).

  ex: `$set a=val;echo $a;ls $a;$a`

  - special variables: term, precmd, cwdcmd, cwd, ignoreof...

- History

  ex: `$history`

- with !

```
ex: $!ls
ex: $!12
ex: $!-4
```

- with ! and modifiers

```
ex: $!ls:s/.c/.h
```

- link with line edition

- Alias
- Line edition:
  - multi line
  - with dynamic rebinding
  - dynamic completion (file, command, contextual...)
- job control (highly appreciated)
- scripting (very long)

.5 Advice

- First advice: make a solid group
  - Be sure that you can really work together (hour, time, characters).
  - Work truly as a group (together and talking about it).
  - Spend lots of time analyzing things at all levels.
  - Confront all your ideas before you jump to realization.
  - Make sure you have understood and that the other members of your group have understood the same thing.
  - Speak to other groups.

- Second advice: move slowly
  - Do not code anything until everything is clear.
  - Do not code anything until you have all your minishells fully functional (for all group members). We even recommend that you completely redo them as a group, just to see how you code together.
  - Make complete working scenarios of your shell. Make tests for what you plan to code. Test all cases you can think of, even the strangest ones (we will test them in the oral defense).
  - Make tests before you start coding.
  - Confront your lists of cases with other groups.
  - Make a clear list of the options that you intend to do, separating the steps.
  - Make a general plan on paper before writing the first line.
  - Test everything during the development. Don't wait for the functionality to be "finished".
  - Don't hesitate to erase parts that seem suspicious, flawed or poorly written (even functional). This will serve you for the rest.
  - Don't do anything that you don't understand completely.
  - Merge it very often (at least 1 to 2 times per week) and code beside each other.

`When your project seems functional, ask other groups to test it...`

.6 Authorized functions

- access
- open
- read
- write
- close
- pipe
- dup
- dup2
- fork
- getpid
- getuid
- geteuid
- getgid
- vfork
- execve
- stat
- lstat
- fstat,
- getsid
- getpgid
- getpgrp
- setpgrp
- setpgid
- setsid
- tcsetpgrp

- tcsetattr
- tcgetpgrp
- tcgetattr
- isatty,
- getpwnam
- getpwent
- getpwuid
- getcwd
- chdir
- opendir
- readdir
- closedir
- glob
- signal
- kill
- wait
- waitpid
- wait3
- wait4
- The printf (and derivates) function
- The string.h and strings.h functions
- errno, malloc, realloc, calloc, free, bzero, memcpy, memcmp, memset, strerror, va_start, va_arg, va_list, va_end, va_copy
- Everything which is related to termcaps is allowed (the window management of libcurses is not part of the termcaps)