`osh` (Simple Shell)

Thông tin đồ án

- Đồ án cho môn Hệ Điều Hành (CSC10007-18CLC2), FIT @ HCMUS
- Nhóm gồm 3 thành viên
  - `18127231`: Đoàn Đình Toàn, @t3bol90
  - `18127185`: Bùi Vũ Hiếu Phụng, @alecmatts
  - `18127221`: Bùi Văn Thiện, @84436

Công việc cần làm

Tính năng	Toàn	Thiện	Phụng
Parse	x	x	
Thực thi lệnh	x		
`&` **(thực thi lệnh dưới nền)**	x		
`<`, `>`, `<<` **(chuyển hướng I/O)**	x	x	
`|` **(pipe)**	x		
Lịch sử lệnh (có thể thay đổi size)			x
`PS1` (shell prompt)		x	x
`cd` (chuyển thư mục)	x		x
`help`		x	
Makefile		x	

*Tất cả các chức năng đều đạt độ hoàn thiện 100%

Hướng thiết kế của đồ án

Ngôn ngữ và môi trường lập trình

Đồ án được lập trình bằng ngôn ngữ `C`,biên dịch bằng `gcc` (sử dụng Makefile) trên môi trường `Ubuntu` (version `18.04` trở lên) với 2 IDE được sử dụng bởi các thành viên trong nhóm là `Visual Studio Code` và `CLion`.

Kịch bản của chương trình

0. Dùng `make` để build chương trình. Nhập `./bin/osh` để chạy chương trình lên.

1. Khi shell bắt đầu chạy, nó sẽ vào một vòng lặp Read-Exec (đọc lệnh - thực thi) cho tới khi nhận được lệnh `exit`
2. Đối với Read: shell sẽ thông qua `parse2()` để đọc-hiểu lệnh
    - Mặc định, `parse2()` nhận nhóm {lệnh + tham số} đầu tiên
    - Nếu có sử dụng op `|` (pipe) hoặc chuyển hướng I/O tương ứng, `parse2()` nhận tiếp nhóm {lệnh + tham số} thứ hai và op đã được đưa vào
    - Nếu có sử dụng op `&` (thực thi dưới nền), `parse2()` trả về op đã sử dụng trong trường hợp này
    - Nếu có nhiều hơn 1 op xuất hiện trong command và/hoặc op đặt ở vị trí không thích hợp: `parse2()` đánh dấu lệnh được đưa là "không hợp lệ/không được hỗ trợ" và ngừng parse
3. Đối với Exec: Shell sẽ dựa vào thông tin đã nhận được ở `parse2()` để fork shell hiện tại ra và, tùy thuộc vào việc lệnh đó có phải là `built-in` của shell hay không, sẽ hành động tương ứng.
    - Nếu như lệnh là `built-in`: đưa lệnh vào một trong các hàm `built_in_*` tương ứng.
    - Nếu không phải là `built-in`:
        - Nếu như lệnh được đánh dấu là "không hợp lệ/không được hỗ trợ": bỏ qua lệnh và báo lỗi
        - Trong các trường hợp còn lại, các nhóm {lệnh + tham số} sẽ được đưa vào một trong các hàm `child_*` tương ứng với op đã phát hiện được trong lệnh.

Mô hình của `parse2()`

`parse2()` có tên như vậy là vì nó là bản cài đặt lại của `parse()` để có thể nhận được các op (`&`, `|`, `<`, `>` và `>>`) mà *không sử dụng* `strtok()`, `strchr()` *hay các công cụ để tokenize/tìm kí tự trong chuỗi khác trong thư viện xử lí C string có sẵn*.

Hướng cài đặt/mô hình của các lệnh

- **Thực thi cơ bản** Mô hình xử lí của chức năng thực thi rất đơn giản. Một process `parent` (process gốc) gọi `fork()` và tạo ra một process `child`, bản thân `parent` dùng `waitpid()` để chờ kết quả trả về của `child` trong khi `child` dùng `execvp()` để thực thi lệnh rồi thoát.

- **Thực thi dưới nền** `&` Tham số `wait` trên tiến trình chính `parent()` có được sau khi parse sẽ quyết định tiến trình này sẽ được chạy dưới nền hay chạy trên mặt. Theo lời của thầy Hòa thì chỉ cần mô phỏng chức năng gửi lệnh xuống nền chứ không quản lí các lệnh được chạy dưới nền (ví dụ: các hàm `jobs`, `fg`) nên mô hình của phần mô phỏng này cũng rất đơn giản. `parent` sẽ gửi `child` một lời nhắn nhủ là *đi rồi về nhớ báo lại*. Sau đó `parent` sử dụng một macro trong C có tên là `WIFEXITED` [1] để kiểm tra xem là `child` có `exit` đúng hay không.

- **`<`, `>`, `>>` (Chuyển hướng I/O)** `dup2()` là một syscall cho phép chúng ta ghi đè một file lên một file khác chỉ cần file description (`fd`). Để thực hiện chức năng chuyển hướng I/O, ta ghi đè file cần chuyển lên `stdin` và `stdout` với `fd` tương ứng là `STDIN_FILENO` và `STDOUT_FILENO` với các mode phù hợp.

- **Đọc từ file:** `child_fromfile()` Mở file bằng lệnh `open` dưới mode `O_RDONLY` (read only) ta sẽ có `fd` của file, giờ chỉ cần dùng `dup2()` để đè lên `stdin`.
- **Ghi vào file:** `child_tofile()` Ở chức năng này nhóm xử lí 2 operator là `>` và `>>`. File vẫn được mở bằng lệnh `open`, nhưng flag chỉ chế độ mở sẽ có thay đổi nhỏ:
  - Đối với `>`, flag là `S_IRWXU` (Read - Write - Execute)
  - Đối với `>>` , flag là `O_WDONLY | O_APPEND` [2]

  Sau đó `dup2()` được dùng để đè file lên `stdout`.

- `|` **(Pipe): Xử lí 2 luồng sử dụng** `child_pipe()` Pipe đúng như tên gọi của nó, một cái ống. Sử dụng `pipe()` để tạo ra cái ống. Mô hình của chúng ta khi xử lí 2 luồng như sau:

  Giả sử hai lệnh nhập vào với cú pháp: `<lệnh A> | <lệnh B>`, mong muốn của chúng ta đó chính là output của lệnh A sẽ là input của lệnh B. Như vậy kết hợp với `dup2()` ta sẽ dẫn được output của A vào đầu bên này ống và đưa đầu bên kia ống để làm input cho B. Trong quá trình này thì nên cân nhắc việc đóng các `fd` không sử dụng lại để tránh gây xung đột.

- **Lịch sử lệnh:** Lịch sử lệnh được cài đặt dưới dạng một mảng tĩnh, tất cả các phần tử được khởi tạo là null. Kích thước mảng lưu lịch sử có thể "thay đổi" được thông qua lệnh `histsize`. Các lệnh được đánh số tăng dần theo thứ tự từ lệnh cũ nhất tới mới nhất.
  - `history`
    - Tồn tại một biến đếm chứa số lệnh đang được lưu trong lịch sử để tránh việc truy cập vào những phần tử mảng đang là null
    - Trong trường hợp có flag `-c` (tức yêu cầu xóa lịch sử), biến đếm số entry trong history sẽ được đặt lại về `0`. [3]
  - `!x` hoặc `!!`
    - Detect dấu `!` để thực hiện lệnh này.
    - Tất cả chữ cái sau dấu `!` tạo thành một mảng kí tự được coi là index cần truy cập tới. Hàm `getIndex()` dùng để chuyển mảng đó thành số để có thể loại trừ được những lệnh không hợp lệ (index chứa kí tự, khoảng trắng,...)
  - `histsize` (thay đổi kích thước lịch sử lệnh): Có 3 trường hợp:
    - (1) Nếu histsize mới được nhập vào bằng với histsize hiện tại, bỏ qua
    - (2) Nếu histsize được nhập vào lớn hơn số lượng lệnh có trong mảng chứa lịch sử, không có sự mất dữ liệu xảy ra. Copy toàn bộ các lệnh có trong mảng cũ sang mảng mới
    - (3) Nếu histsize được nhập vào nhỏ hơn số lượng lệnh đang có trong mảng chứa lịch sử, sự mất mát dữ liệu xảy ra, các lệnh mới hơn sẽ được giữ lại và được copy sang mảng mới

    Ở trường hợp (2) và (3), cần tạo một mảng lịch sử mới. Sau khi sửa đổi nội dung mảng mới này, trỏ con trỏ của mảng cũ đến mảng mới rồi free mảng mới.

    Nếu như lệnh được đánh dấu là "không hợp lệ/không được hỗ trợ": bỏ qua lệnh và báo lỗi

- **Các tính năng linh tinh**
  - `PS1` **(Shell prompt)** Đây chỉ là một string để hiển thị trước dấu nhắc.

- `cd` **(Thay đổi thư mục làm việc)** Việc thay đổi thư mục hiện tại được thực hiện bằng `chdir()` được cung cấp bởi `<unistd.h>`
- `help` **(Hướng dẫn cho các lệnh có sẵn)** Các help messages được cài đặt dưới dạng `static` string. Hàm help chỉ thực hiện một phép so sánh cơ bản trên tham số nhận được (lệnh cụ thể để tra cứu) nếu cần, sau đó hiện help message tương ứng nếu lệnh tồn tại, hoặc thông báo lỗi nếu lệnh không tồn tại.

Nhận xét của nhóm

Về lựa chọn C + Makefile

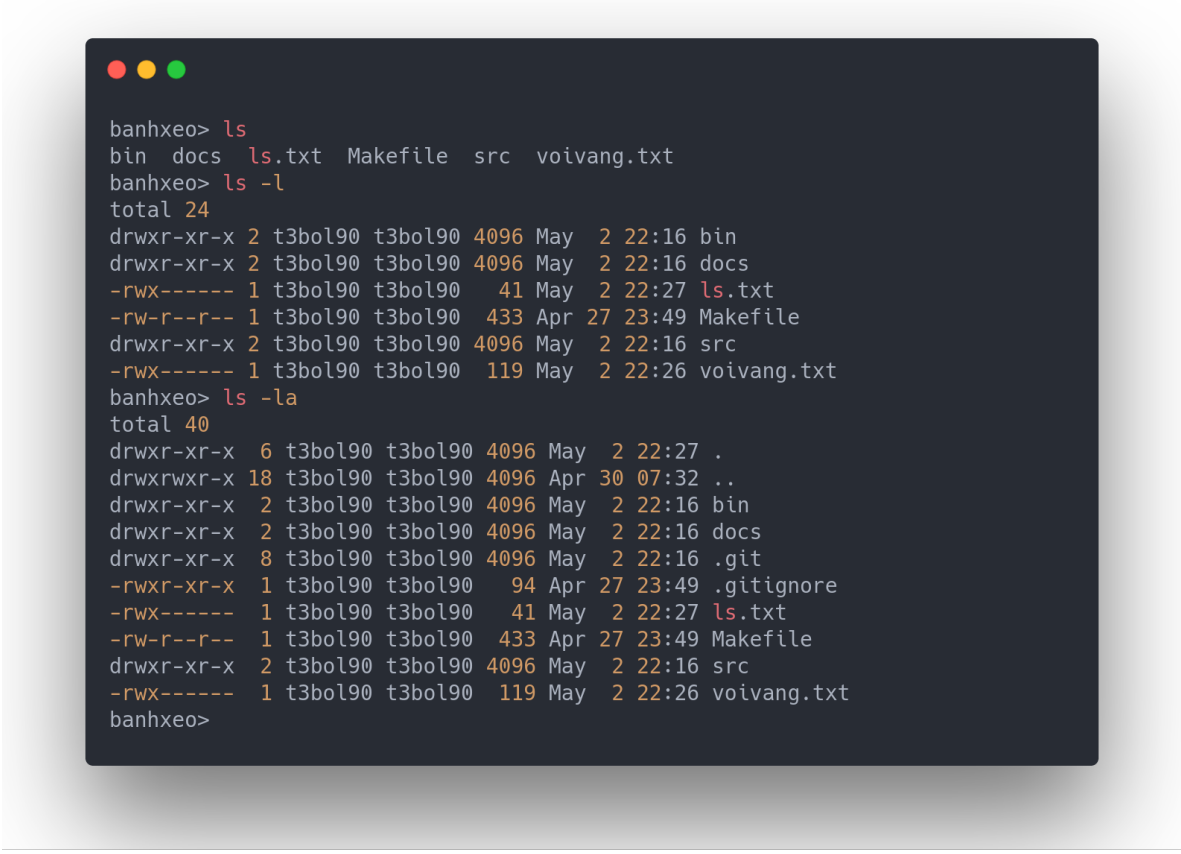
- C là ngôn ngữ được dùng để tạo ra Linux kernel ở thuở sơ khai. Đến ngày nay, bản thân Linux và một số công cụ chạy trên nó được viết bằng C.
- > Nhóm thích sự thử thách bản thân với `pointer`, `malloc`,... những thứ mà `C++` hay những ngôn ngữ khác không có được hoặc khó để trải nghiệm. Đồ án này đơn giản là một trải nghiệm, thế thôi :>
  >
  > -- Toàn Đoàn, lead nhóm
- Makefile là một công cụ truyền thống hỗ trợ việc tự động các tác vụ khi compile chương trình trên Linux/*nix. Makefile ngắn gọn (nhưng không dễ đọc :<) và tương đối dễ sử dụng hơn CMake; ta có thể xem nó như một cái script (nhưng với cú pháp không dễ đọc/viết :<)

Về lớp lệnh built-in

Các lệnh built-in của shell hiện chỉ mang tính sơ khai và phục vụ một số tính năng phụ (như thay đổi kích thước lịch sử lệnh sử dụng `histsize`, xem hướng dẫn sử dụng `histsize`, v.v.). Lớp lệnh này có thể phát triển thêm nếu có thời gian mở rộng.

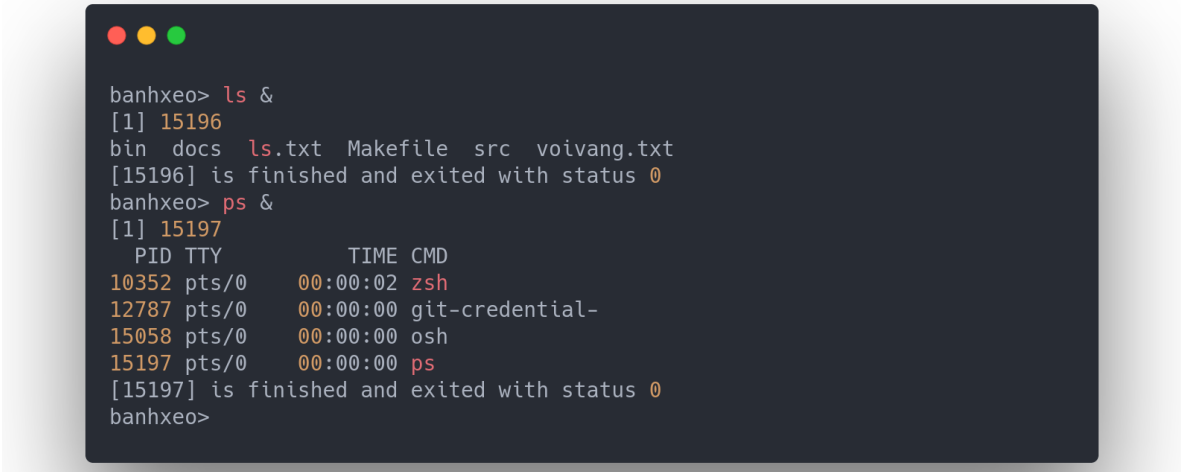
Các testcase

Thực thi lệnh



```
banhxeo> ls
bin  docs  ls.txt  Makefile  src  voivang.txt
banhxeo> ls -l
total 24
drwxr-xr-x 2 t3bol90 t3bol90 4096 May  2 22:16 bin
drwxr-xr-x 2 t3bol90 t3bol90 4096 May  2 22:16 docs
-rwx------ 1 t3bol90 t3bol90   41 May  2 22:27 ls.txt
-rw-r--r-- 1 t3bol90 t3bol90  433 Apr 27 23:49 Makefile
drwxr-xr-x 2 t3bol90 t3bol90 4096 May  2 22:16 src
-rwx------ 1 t3bol90 t3bol90  119 May  2 22:26 voivang.txt
banhxeo> ls -la
total 40
drwxr-xr-x  6 t3bol90 t3bol90 4096 May  2 22:27 .
drwxrwxr-x 18 t3bol90 t3bol90 4096 Apr 30 07:32 ..
drwxr-xr-x  2 t3bol90 t3bol90 4096 May  2 22:16 bin
drwxr-xr-x  2 t3bol90 t3bol90 4096 May  2 22:16 docs
drwxr-xr-x  8 t3bol90 t3bol90 4096 May  2 22:16 .git
-rwxr-xr-x  1 t3bol90 t3bol90   94 Apr 27 23:49 .gitignore
-rwx------  1 t3bol90 t3bol90   41 May  2 22:27 ls.txt
-rw-r--r--  1 t3bol90 t3bol90  433 Apr 27 23:49 Makefile
drwxr-xr-x  2 t3bol90 t3bol90 4096 May  2 22:16 src
-rwx------  1 t3bol90 t3bol90  119 May  2 22:26 voivang.txt
banhxeo>
```

Thực thi lệnh dưới nền



```
banhxeo> ls &
[1] 15196
bin  docs  ls.txt  Makefile  src  voivang.txt
[15196] is finished and exited with status 0
banhxeo> ps &
[1] 15197
  PID TTY          TIME CMD
10352 pts/0    00:00:02 zsh
12787 pts/0    00:00:00 git-credential-
15058 pts/0    00:00:00 osh
15197 pts/0    00:00:00 ps
[15197] is finished and exited with status 0
banhxeo>
```

Điều hướng nhập xuất

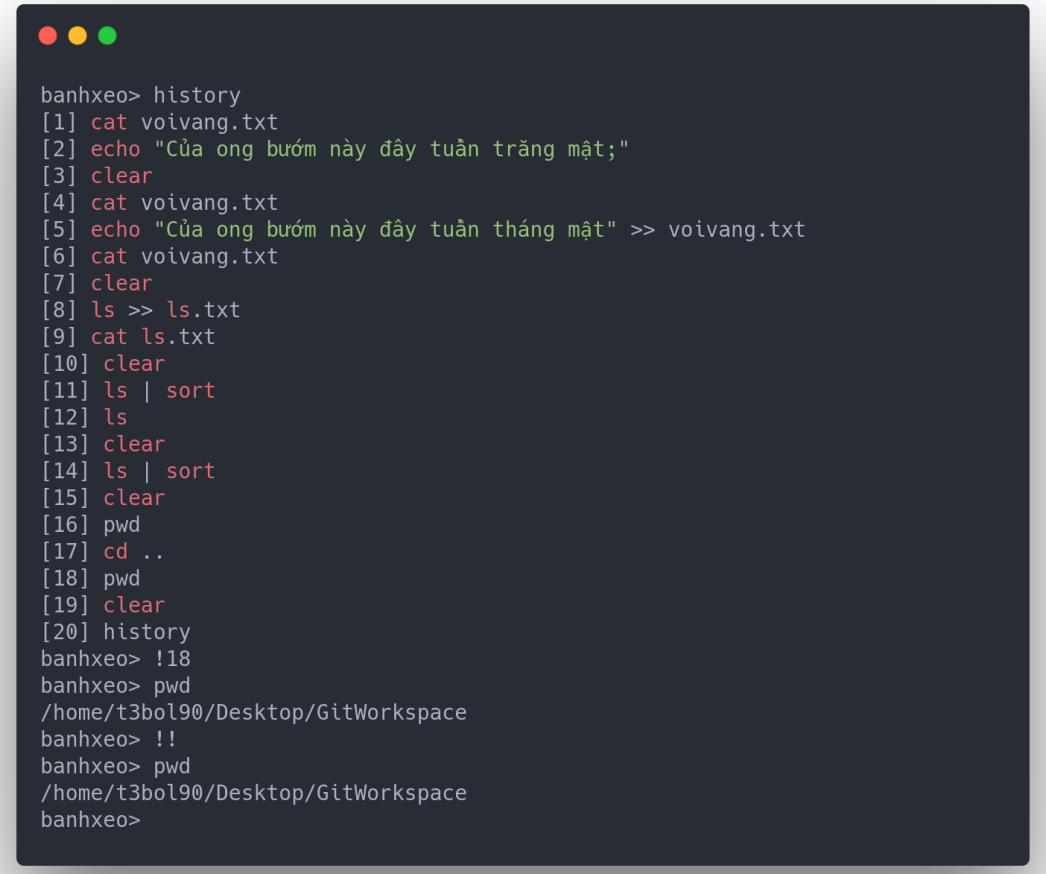
```
banhxeo> ls > ls.txt
banhxeo> cat ls.txt
bin
docs
ls.txt
Makefile
src
voivang.txt
banhxeo>
```

```
banhxeo> cat voivang.txt
Tôi muốn tắt nắng đi
Cho màu đừng nhạt mất;
Tôi muốn buộc gió lại
Cho hương đừng bay đi.
banhxeo> sort < voivang.txt
Cho hương đừng bay đi.
Cho màu đừng nhạt mất;
Tôi muốn buộc gió lại
Tôi muốn tắt nắng đi
banhxeo>
```

Thực thi với pipe

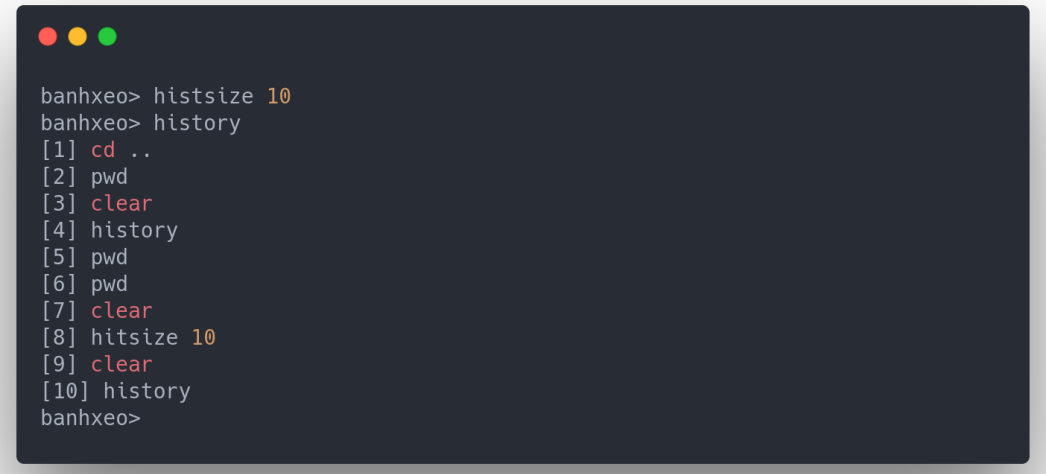
```
banhxeo> ls | sort
bin
docs
ls.txt
Makefile
src
voivang.txt
banhxeo>
```

Tra cứu lịch sử



```
banhxeo> history
[1] cat voivang.txt
[2] echo "Của ong bướm này đây tuần trăng mật;"
[3] clear
[4] cat voivang.txt
[5] echo "Của ong bướm này đây tuần tháng mật" >> voivang.txt
[6] cat voivang.txt
[7] clear
[8] ls >> ls.txt
[9] cat ls.txt
[10] clear
[11] ls | sort
[12] ls
[13] clear
[14] ls | sort
[15] clear
[16] pwd
[17] cd ..
[18] pwd
[19] clear
[20] history
banhxeo> !18
banhxeo> pwd
/home/t3bol90/Desktop/GitWorkspace
banhxeo> !!
banhxeo> pwd
/home/t3bol90/Desktop/GitWorkspace
banhxeo>
```

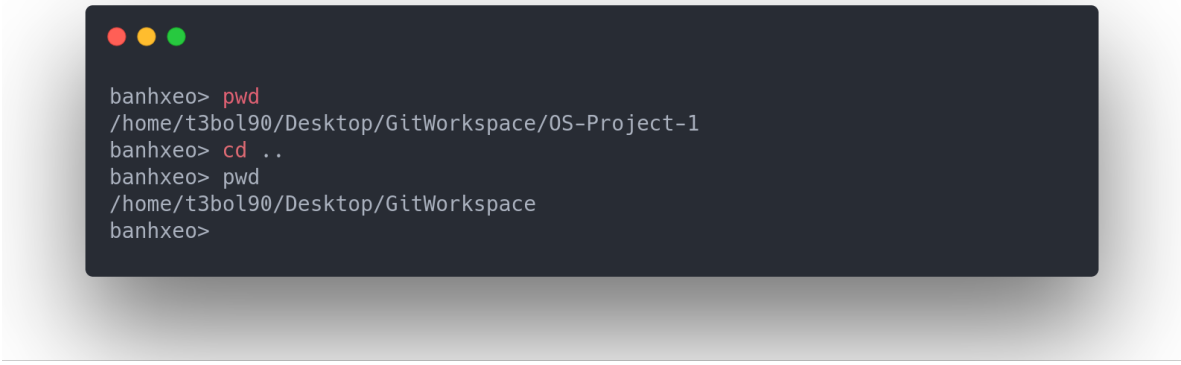
Thay đổi kích thước danh sách lịch sử lệnh



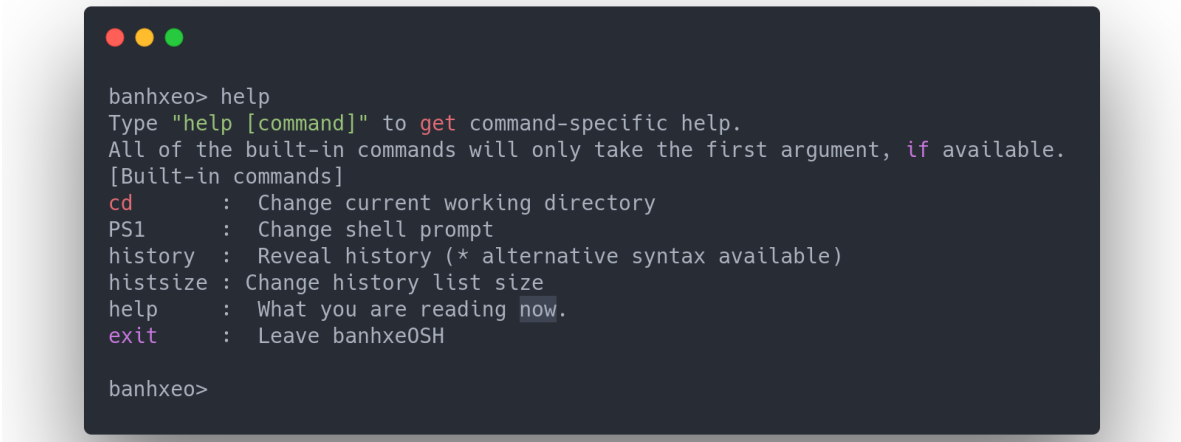
```
banhxeo> histsize 10
banhxeo> history
[1] cd ..
[2] pwd
[3] clear
[4] history
[5] pwd
[6] pwd
[7] clear
[8] hitsize 10
[9] clear
[10] history
banhxeo>
```

Thay đổi tên shell

Chuyển thư mục làm việc



Help (Hướng dẫn sử dụng cho các lệnh built-in)



Lời cảm ơn:

Để hoàn thiện được đồ án này, không thể nhắc tới sự hướng dẫn từ những thầy cô ở HCMUS. Gửi lời cảm ơn chân thành đến:

- Thầy **Lê Quốc Hòa**, giảng viên phụ trách bộ môn Hệ Điều Hành.

- Cô **Chung Thùy Linh**, giảng viên hướng dẫn thực hành và là người trực tiếp hướng dẫn đồ án.

Một lời cảm ơn nữa đến những người đồng đội đã cùng nhau thực hiện và hoàn thiện đồ án này cho tới thời điểm cuối cùng.

> Chúc mọi người có nhiều sức khỏe và thành công trong công việc.

Tài liệu tham khảo:

Linux's man page ở dạng web và dạng `CLI` với lệnh `man <lệnh cần tra cứu>.`

Footnotes

---

1. Thực ra còn có một macro khác có tên là `WIFSIGNALED` để kiểm tra xem child có bị terminate bởi một signal nào đó send tới không (để tránh trường hợp zombie process). Nhưng đồ án này hiện chưa có handle signal (mô tả đồ án cũng không yêu cầu). Nên nhóm sẽ bỏ qua vấn đề này nói riêng và signal handling nói chung. ↩

2. Bài học của nhóm: đây không phải là C++. `open(2)` nêu rõ flag chỉ chế độ mở **phải** đi kèm một trong 3 mode: `O_RDONLY` (chỉ đọc), `O_WRONLY` (chỉ ghi), hoặc `O_RDWR` (đọc và ghi). ↩

3. Mỗi một entry mới sẽ được ghi đè bắt đầu từ `history_count` lúc đó (tức bắt đầu từ `[0]`, ở đầu list), nên cách làm này dù quick-and-dirty nhưng có lẽ an toàn (entry cũ không thể truy cập được). Ta không cần phải `free()` hay `allocate` lại từ đầu vì cơ bản khi người dùng gõ lệnh mới sẽ lưu đè lên vùng nhớ cũ - thứ mà ta đã không cần quan tâm tới nữa. ↩