(/)

0x16. C - Simple Shell

C | Group project | Syscall

By: Julien Barbier

Weight: 10

Project to be done in teams of 2 people (your team: Rachid Amhouch, Saliha Brik)

Project over - took place from May 10, 2023 4:00 AM to May 25, 2023 4:00 AM

An auto review will be launched at the deadline

In a nutshell…

- **Contribution:** 100.0%
- **Auto QA review:** 55.0/55 mandatory & 76.0/76 optional
- **Altogether: 200.0%**
  - Mandatory: 100.0%
  - Optional: 100.0%
  - Contribution: 100.0%
  - Calculation: 100.0% * (100.0% + (100.0% * 100.0%)) == **200.0%**

Concepts

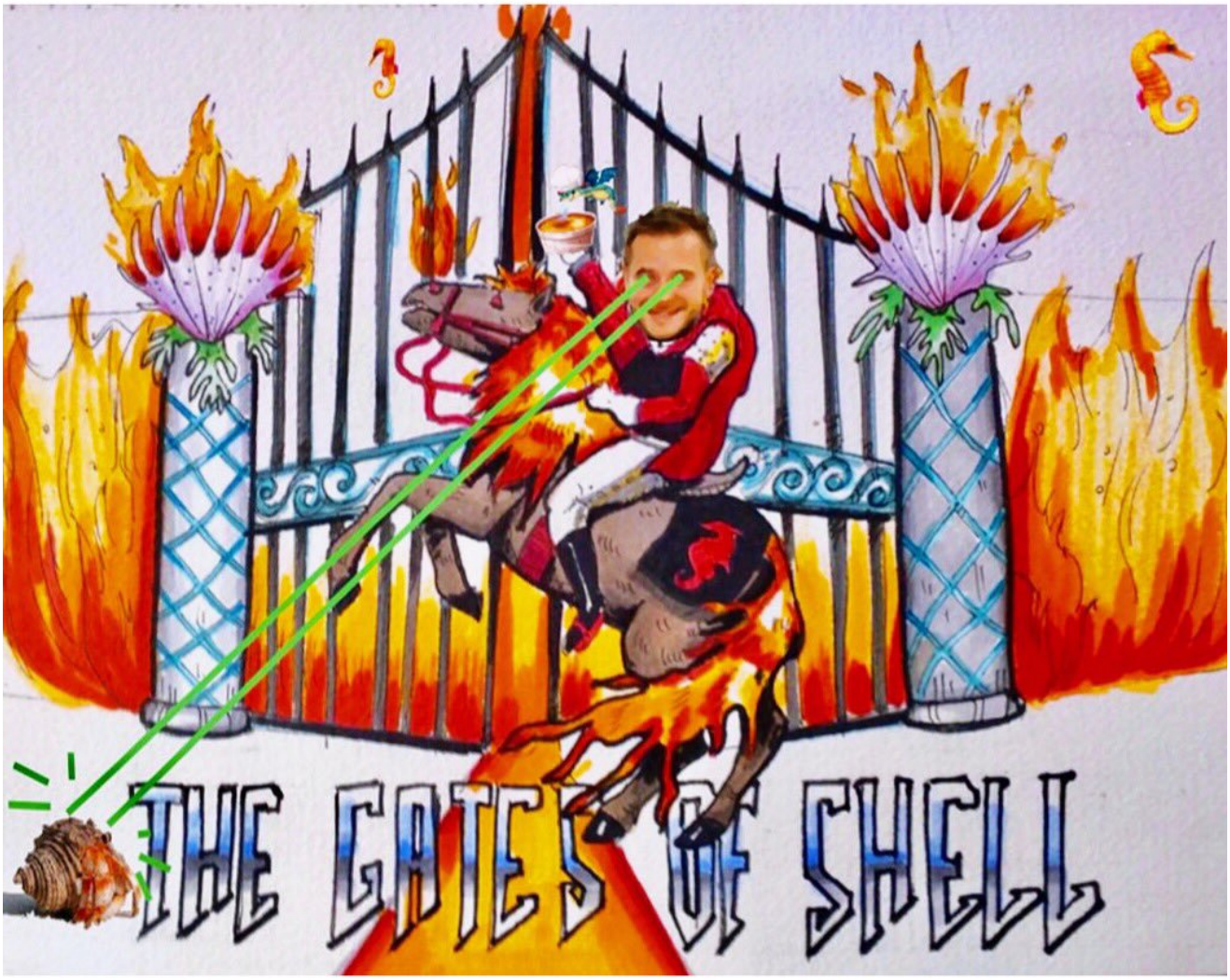
For this project, we expect you to look at these concepts:

- Everything you need to know to start coding your own shell (/concepts/64)
- Approaching a Project (/concepts/350)

Background Context

Write a simple UNIX command interpreter.

(/)



^ *"The Gates of Shell", by Spencer Cheng (/rltoken/AtYRSM03vJDrko9xHodxFQ), featuring Julien Barbier (/rltoken/-ezXgcyfhc8qU1DeUlnLUA)*

Important message from Julien

It's time for the famous Simple Shell project. This is one of the most anticipated project and also one that will challenge you a lot about everything you have learn so far:

- Basics of programming
- Basics of C
- Basics of thinking like an engineer
- Group work
- and Learning how to learn

I would like to take this moment to remind you about a few important things.

First, remember the framework. If you do not know it by heart already, it is probably a good idea to read it again: https://intranet.alxswe.com/concepts/559 (/rltoken/a08_c01OP1XHY3awtkdFRA)

Note that there is no point in this framework that says it is ok to look at code from other people. It is not allowed to look at other people's code, either other students or online articles or videos. At ALX SE we do not copy solutions and we do not look at it when we start a project.

In the context of learning (some of these will no longer be true when you work):

- NEVER copy any code, never look at solution (and never give any solution to your friends, you are not helping them by doing so)
- ALWAYS write code alone from scratch after you get help to check that you have actually understood. If you can not do it, you have not understood enough, and need to study more. Do not rewrite code from memory, but from understanding.

I saw some of you sharing resources with each other already. Tutorials on how to do the shell step by step with all the code associated with these, or even video and documents with the solution without even any explanation. This is not the right way to learn. Please do not be tempted by these links. They will only push you to take shortcuts and / or cheat. And trust me, you will be caught. Kimba (/rltoken/3nocfYiMMxjbhlMllUqLxg) is not a joke and he is here to remind you why you are here.

While we encourage the use of ChatGPT and co in the framework (also, not right away, but at the right step, see framework), it is important to understand that the same rules apply to these AI tools (again, in the context of learning. When you will work it will be completely different, but context matters). At no point does it say that you are allowed to use copilot or ChatGPT to code the solution. If you do, you will get 200% (for a few hours), understand 0, learn 0, and you will be caught for cheating 100%, and then your score for both you and your partner will be 0%. If you don't get how to use ChatGPT and other AI tools in the context of learning, simply do not use them.

The reality is that at this point of the program, if you have not cheated before, you have everything you need to complete the project with what you have learned + the page "Everything you need to know to start coding your own shell" https://intranet.alxswe.com/concepts/64 (/rltoken/e6Nw3W01-33JDxlCyKX-Kw)

Actually, you do not even need to open Google once. Focus on your whiteboarding, and everything will fall in place. Remember, at ALX SE you never learn the solution, you learn how to walk toward the solution. You learn to create the tutorial, so if you follow one, you are looking at the solution, you are taking a very serious shortcut that will undermine your learning.

Last thing about the framework. Note that the first thing to do is "0. Read". Every detail counts. Make sure you read and test everything.

The shell project is a group project. That means you will be paired with someone. You already did this with printf, so please apply everything you have learned from the printf experience here. A quick reminder, that a group project is NOT:

- I do nothing and cross fingers for my partner to do everything so I can have a good score
- I do everything because I am so much better than my partner and I don't care about them

A group project at ALX SE is a project that both of you are responsible for. Everything anyone pushes to Github is the responsibility of both partners. It is not ok to say later "I didn't cheat it's my partner I didn't know they didn't tell me".

So you are supposed to work TOGETHER. And you should both understand every single line of code that any of you pushes. Here is a link for you to read about pair programming: https://intranet.alxswe.com/concepts/121 (/rltoken/G52zDoV1f2dmmMl3ngchyw)

If you plan on not working on the shell project (or if at any point in time you can't), it is your responsibility to tell both the staff and your partner so that they can find another partner who will work with them asap.

If your group gets caught for plagiarism we will not tolerate “I didn’t do anything, so I should not be flagged”. Yes you should be flagged, because you are someone who doesn’t care about others and thought it was ok to let your partner down and to maybe get the score without doing anything.

The shell is an incredibly cool project. GL HF!

Julien

Resources

Read or watch:

- Unix shell (/rltoken/f0YU9TAhniMXWlSXtb64Yw)
- Thompson shell (/rltoken/7LJOp2qP7qHUcsOK2-F3qA)
- Ken Thompson (/rltoken/wTSu31ZP1f7fFTJFgRQC7w)
- **Everything you need to know to start coding your own shell** concept page

man or help:

- `sh` (*Run `sh` as well*)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/9LNz86CtOTos9oL3zxIO3A), **without the help of Google**:

General

- Who designed and implemented the original Unix operating system
- Who wrote the first version of the UNIX shell
- Who invented the B programming language (the direct predecessor to the C programming language)
- Who is Ken Thompson
- How does a shell work
- What is a pid and a ppid
- How to manipulate the environment of the current process
- What is the difference between a function and a system call
- How to create processes
- What are the three prototypes of `main`
- How does the shell use the `PATH` to find the programs
- How to execute another program with the `execve` system call
- How to suspend the execution of a process until one of its children terminates
- What is `EOF` / “end-of-file”?

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else’s work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

General

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be compiled on Ubuntu 20.04 LTS using `gcc`, using the options `-Wall -Werror -Wextra -pedantic -std=gnu89`
- All your files should end with a new line
- A `README.md` file, at the root of the folder of the project is mandatory
- Your code should use the `Betty` style. It will be checked using betty-style.pl (https://github.com/alx-tools/Betty/blob/master/betty-style.pl) and betty-doc.pl (https://github.com/alx-tools/Betty/blob/master/betty-doc.pl)
- Your shell should not have any memory leaks
- No more than 5 functions per file
- All your header files should be include guarded
- Use system calls only when you need to (why? (/rltoken/EU7B1PTSy14lNnZEShpobQ))
- Write a `README` with the description of your project
- You should have an `AUTHORS` file at the root of your repository, listing all individuals having contributed content to the repository. Format, see Docker (/rltoken/UL8J3kgl7HBK_Z9iBL3JFg)

GitHub

**There should be one project repository per group. If you and your partner have a repository with the same name in both your accounts, you risk a 0% score. Add your partner as a collaborator. **

More Info

Output

- Unless specified otherwise, your program **must have the exact same output** as `sh` (`/bin/sh`) as well as the exact same error output.
- The only difference is when you print an error, the name of the program must be equivalent to your `argv[0]` (See below)

Example of error with `sh`:

```
$ echo "qwerty" | /bin/sh
/bin/sh: 1: qwerty: not found
$ echo "qwerty" | /bin/../bin/sh
/bin/../bin/sh: 1: qwerty: not found
$
```

Same error with your program `hsh`:

```
$ echo "qwerty" | ./hsh
./hsh: 1: qwerty: not found
$ echo "qwerty" | ./././hsh
./././hsh: 1: qwerty: not found
$
```

List of allowed functions and system calls

(/)

- `access` (man 2 access)
- `chdir` (man 2 chdir)
- `close` (man 2 close)
- `closedir` (man 3 closedir)
- `execve` (man 2 execve)
- `exit` (man 3 exit)
- `_exit` (man 2 _exit)
- `fflush` (man 3 fflush)
- `fork` (man 2 fork)
- `free` (man 3 free)
- `getcwd` (man 3 getcwd)
- `getline` (man 3 getline)
- `getpid` (man 2 getpid)
- `isatty` (man 3 isatty)
- `kill` (man 2 kill)
- `malloc` (man 3 malloc)
- `open` (man 2 open)
- `opendir` (man 3 opendir)
- `perror` (man 3 perror)
- `read` (man 2 read)
- `readdir` (man 3 readdir)
- `signal` (man 2 signal)
- `stat` (__xstat) (man 2 stat)
- `lstat` (__lxstat) (man 2 lstat)
- `fstat` (__fxstat) (man 2 fstat)
- `strtok` (man 3 strtok)
- `wait` (man 2 wait)
- `waitpid` (man 2 waitpid)
- `wait3` (man 2 wait3)
- `wait4` (man 2 wait4)
- `write` (man 2 write)

Compilation

Your shell will be compiled this way:

```
gcc -Wall -Werror -Wextra -pedantic -std=gnu89 *.c -o hsh
```

Testing

Your shell should work like this in interactive mode:

```
$ ./hsh
($) /bin/ls
hsh main.c shell.c
($)
($) exit
$
```

But also in non-interactive mode:

```
$ echo "/bin/ls" | ./hsh
hsh main.c shell.c test_ls_2
$
$ cat test_ls_2
/bin/ls
/bin/ls
$
$ cat test_ls_2 | ./hsh
hsh main.c shell.c test_ls_2
hsh main.c shell.c test_ls_2
$
```

Checks

The Checker will be released at the end of the project (1-2 days before the deadline). We **strongly** encourage the entire class to work together to create a suite of checks covering both regular tests and edge cases for each task. See task `8. Test suite`.

Tasks

0. Betty would be proud

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a beautiful code that passes the Betty checks

Repo:

- GitHub repository: `simple_shell`

Done! | Help | Check your code | >_ Get a sandbox | QA Review

1. Simple shell 0.1

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

(/)

Write a UNIX command line interpreter.

- Usage: `simple_shell`

Your Shell should:

- Display a prompt and wait for the user to type a command. A command line always ends with a new line.
- The prompt is displayed again each time a command has been executed.
- The command lines are simple, no semicolons, no pipes, no redirections or any other advanced features.
- The command lines are made only of one word. No arguments will be passed to programs.
- If an executable cannot be found, print an error message and display the prompt again.
- Handle errors.
- You have to handle the “end of file” condition (`Ctrl+D`)

You don’t have to:

- use the `PATH`
- implement built-ins
- handle special characters : `"`, `'`, `` ` ``, `\`, `*`, `&`, `#`
- be able to move the cursor
- handle commands with arguments

`execve` *will be the core part of your Shell, don’t forget to pass the environ to it...*

```
julien@ubuntu:~/shell$ ./shell
#cisfun$ ls
./shell: No such file or directory
#cisfun$ /bin/ls
barbie_j       env-main.c  exec.c  fork.c  pid.c  ppid.c    prompt   prompt.c  sh
ell.c  stat.c         wait
env-environ.c  exec    fork    mypid   ppid   printenv  promptc  shell     stat t
est_scripting.sh  wait.c
#cisfun$ /bin/ls -l
./shell: No such file or directory
#cisfun$ ^[[D^[[D^[[D
./shell: No such file or directory
#cisfun$ ^[[C^[[C^[[C^[[C
./shell: No such file or directory
#cisfun$ exit
./shell: No such file or directory
#cisfun$ ^C
julien@ubuntu:~/shell$ echo "/bin/ls" | ./shell
barbie_j       env-main.c  exec.c  fork.c  pid.c  ppid.c    prompt   prompt.c  sh
ell.c  stat.c         wait
env-environ.c  exec    fork    mypid   ppid   printenv  promptc  shell     stat t
est_scripting.sh  wait.c
#cisfun$ julien@ubuntu:~/shell$
```

Repo:

- GitHub repository: `simple_shell`

Done! | Help | Check your code | >_ Get a sandbox | QA Review

2. Simple shell 0.2

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 0.1 +

- Handle command lines with arguments

Repo:

- GitHub repository: `simple_shell`

Done! | Help | Check your code | >_ Get a sandbox | QA Review

3. Simple shell 0.3

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 0.2 +

- Handle the `PATH`
- `fork` must not be called if the command doesn't exist

```
julien@ubuntu:~/shell$ ./shell_0.3
:) /bin/ls
barbie_j       env-main.c  exec.c  fork.c  pid.c  ppid.c    prompt   prompt.c  sh
ell_0.3  stat    test_scripting.sh  wait.c
env-environ.c  exec    fork    mypid   ppid   printenv  promptc  shell     shell.
c    stat.c  wait
:) ls
barbie_j       env-main.c  exec.c  fork.c  pid.c  ppid.c    prompt   prompt.c  sh
ell_0.3  stat    test_scripting.sh  wait.c
env-environ.c  exec    fork    mypid   ppid   printenv  promptc  shell     shell.
c    stat.c  wait
:) ls -l /tmp
total 20
-rw------- 1 julien julien    0 Dec  5 12:09 config-err-aAMZrR
drwx------ 3 root   root   4096 Dec  5 12:09 systemd-private-062a0eca7f2a44349733
e78cb4abdff4-colord.service-V7DUzr
drwx------ 3 root   root   4096 Dec  5 12:09 systemd-private-062a0eca7f2a44349733
e78cb4abdff4-rtkit-daemon.service-ANGvoV
drwx------ 3 root   root   4096 Dec  5 12:07 systemd-private-062a0eca7f2a44349733
e78cb4abdff4-systemd-timesyncd.service-CdXUtH
-rw-rw-r-- 1 julien julien    0 Dec  5 12:09 unity_support_test.0
:) ^C
julien@ubuntu:~/shell$
```

Repo:

- GitHub repository: `simple_shell`

Done! | Help | Check your code | >_ Get a sandbox | QA Review

4. Simple shell 0.4

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 0.3 +

- Implement the `exit` built-in, that exits the shell
- Usage: `exit`
- You don't have to handle any argument to the built-in `exit`

Repo:

- GitHub repository: `simple_shell`

Done! | Help | Check your code | >_ Get a sandbox | QA Review

5. Simple shell 1.0

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 0.4 +

- Implement the `env` **built-in**, that prints the current environment

```
julien@ubuntu:~/shell$ ./simple_shell
$ env
USER=julien
LANGUAGE=en_US
SESSION=ubuntu
COMPIZ_CONFIG_PROFILE=ubuntu
SHLVL=1
HOME=/home/julien
C_IS=Fun_:)
DESKTOP_SESSION=ubuntu
LOGNAME=julien
TERM=xterm-256color
PATH=/home/julien/bin:/home/julien/.local/bin:/usr/local/sbin:/usr/local/bin:/us
r/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
DISPLAY=:0
$ exit
julien@ubuntu:~/shell$
```

Repo:

- GitHub repository: `simple_shell`

Done! | Help | Check your code | >_ Get a sandbox | QA Review

6. Simple shell 0.1.1

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 0.1 +

- Write your own `getline` function
- Use a buffer to read many chars at once and call the least possible the `read` system call
- You will need to use `static` variables
- You are not allowed to use `getline`

You don't have to:

- be able to move the cursor

Repo:

- GitHub repository: `simple_shell`

(/)

Done! | Help | Check your code | >_ Get a sandbox | QA Review

7. Simple shell 0.2.1

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 0.2 +

- You are not allowed to use `strtok`

Repo:

- GitHub repository: `simple_shell`

Done! | Help | Check your code | >_ Get a sandbox | QA Review

8. Simple shell 0.4.1

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 0.4 +

- handle arguments for the built-in `exit`
- Usage: `exit status`, where `status` is an integer used to exit the shell

```
julien@ubuntu:~/shell$ ./shell_0.4.1
$ exit 98
julien@ubuntu:~/shell$ echo $?
98
julien@ubuntu:~/shell$
```

Repo:

- GitHub repository: `simple_shell`

Done! | Help | Check your code | >_ Get a sandbox | QA Review

9. setenv, unsetenv

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Score: 100.0% (*Checks completed: 100.0%*)

(/)

Simple shell 1.0 +

Implement the `setenv` and `unsetenv` builtin commands

- `setenv`
  - Initialize a new environment variable, or modify an existing one
  - Command syntax: `setenv VARIABLE VALUE`
  - Should print something on stderr on failure
- `unsetenv`
  - Remove an environment variable
  - Command syntax: `unsetenv VARIABLE`
  - Should print something on stderr on failure

Repo:

- GitHub repository: `simple_shell`

☑ Done! | Help | Check your code | >_ Get a sandbox | QA Review

10. cd

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 1.0 +

Implement the builtin command `cd`:

- Changes the current directory of the process.
- Command syntax: `cd [DIRECTORY]`
- If no argument is given to `cd` the command must be interpreted like `cd $HOME`
- You have to handle the command `cd -`
- You have to update the environment variable `PWD` when you change directory

`man chdir`, `man getcwd`

Repo:

- GitHub repository: `simple_shell`

☑ Done! | Help | Check your code | >_ Get a sandbox | QA Review

11. ;

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 1.0 +
(/)

- Handle the commands separator `;`

```
alex@~$ ls /var ; ls /var
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
alex@~$ ls /hbtn ; ls /var
ls: cannot access /hbtn: No such file or directory
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
alex@~$ ls /var ; ls /hbtn
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
ls: cannot access /hbtn: No such file or directory
alex@~$ ls /var ; ls /hbtn ; ls /var ; ls /var
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
ls: cannot access /hbtn: No such file or directory
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
alex@~$
```

Repo:

- GitHub repository: `simple_shell`

Done! | Help | Check your code | >_ Get a sandbox | QA Review

12. && and ||

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 1.0 +

- Handle the `&&` and `||` shell logical operators

(/)

```
alex@~$ ls /var && ls /var
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
alex@~$ ls /hbtn && ls /var
ls: cannot access /hbtn: No such file or directory
alex@~$ ls /var && ls /var && ls /var && ls /hbtn
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
ls: cannot access /hbtn: No such file or directory
alex@~$ ls /var && ls /var && ls /var && ls /hbtn && ls /hbtn
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
ls: cannot access /hbtn: No such file or directory
alex@~$
alex@~$ ls /var || ls /var
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
alex@~$ ls /hbtn || ls /var
ls: cannot access /hbtn: No such file or directory
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
alex@~$ ls /hbtn || ls /hbtn || ls /hbtn || ls /var
ls: cannot access /hbtn: No such file or directory
ls: cannot access /hbtn: No such file or directory
ls: cannot access /hbtn: No such file or directory
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
alex@~$ ls /hbtn || ls /hbtn || ls /hbtn || ls /var || ls /var
ls: cannot access /hbtn: No such file or directory
ls: cannot access /hbtn: No such file or directory
ls: cannot access /hbtn: No such file or directory
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  spool  tmp
alex@~$
```

Repo:

- GitHub repository: `simple_shell`

☑ Done! | Help | Check your code | >_ Get a sandbox | QA Review

13. alias

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 1.0 +

- Implement the `alias` builtin command
- Usage: `alias [name[='value'] ...]`
  - `alias` : Prints a list of all aliases, one per line, in the form `name='value'`
  - `alias name [name2 ...]` : Prints the aliases `name` , `name2` , etc 1 per line, in the form `name='value'`

(/)

- `alias name='value' [...]` : Defines an alias for each `name` whose `value` is given. If `name` is already an alias, replaces its value with `value`

Repo:

- GitHub repository: `simple_shell`

Done! | Help | Check your code | >_ Get a sandbox | QA Review

14. Variables

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 1.0 +

- Handle variables replacement
- Handle the `$?` variable
- Handle the `$$` variable

```
julien@ubuntu:~/shell$ ./hsh
$ ls /var
backups  cache  crash  lib  local  lock  log  mail  metrics  opt  run  snap  spoo
l  tmp
$ echo $?
0
$ echo $$
5104
$ echo $PATH
/home/julien/bin:/home/julien/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbi
n:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
$ exit
julien@ubuntu:~/shell$
```

Repo:

- GitHub repository: `simple_shell`

Done! | Help | Check your code | >_ Get a sandbox | QA Review

15. Comments

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 1.0 +

- Handle comments (`#`)

```
julien@ubuntu:~/shell$ sh
$ echo $$ # ls -la
5114
$ exit
julien@ubuntu:~/shell$
```

Repo:

- GitHub repository: `simple_shell`

Done! | Help | Check your code | >_ Get a sandbox | QA Review

16. File as input

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Simple shell 1.0 +

- Usage: `simple_shell [filename]`
- Your shell can take a file as a command line argument
- The file contains all the commands that your shell should run before exiting
- The file should contain one command per line
- In this mode, the shell should not print a prompt and should not read from `stdin`

Repo:

- GitHub repository: `simple_shell`

Done! | Help | Check your code | >_ Get a sandbox | QA Review