**SimpleShell\_Project 2\_Report**

**Group members:**

* **1751064 Nguyen Hoang Gia**
* **1751063 Pham Bao Duy**

**GitHub Repository link:**

[**https://github.com/nhgia/SimpleShell**](https://github.com/nhgia/SimpleShell)

1. **README.md**

### Prerequisite

* Ubuntu / Linux. (Recommended Ubuntu 18.04)
* Download or clone the project.
* Install essential components by running:

sudo apt-get install build-essential

* Change directory to the project folder.

### Install: Compile the C file into output and runnable

* Compile through Makefile, run

make

* Run the shell

./simple-shell

### SimpleShell commands and features

#### 1. Single command

* List sub-folder & sub-item

ls

* Current directory

pwd

* Ping website

ping -i 5 -w 10 -W 10 google.com

* ...

#### 2. Check history of run commands

* Using double exclamation mark "!!" for checking history

!!

#### 3. Concurrence commands

* Add an ampersand mark after the command with whitespace required.

ping -i 5 -w 10 -W 10 google.com &

#### 4. I/O redirection

* Redirect the output of a command to a file and input from a file to a command.

ls > myFile.txt

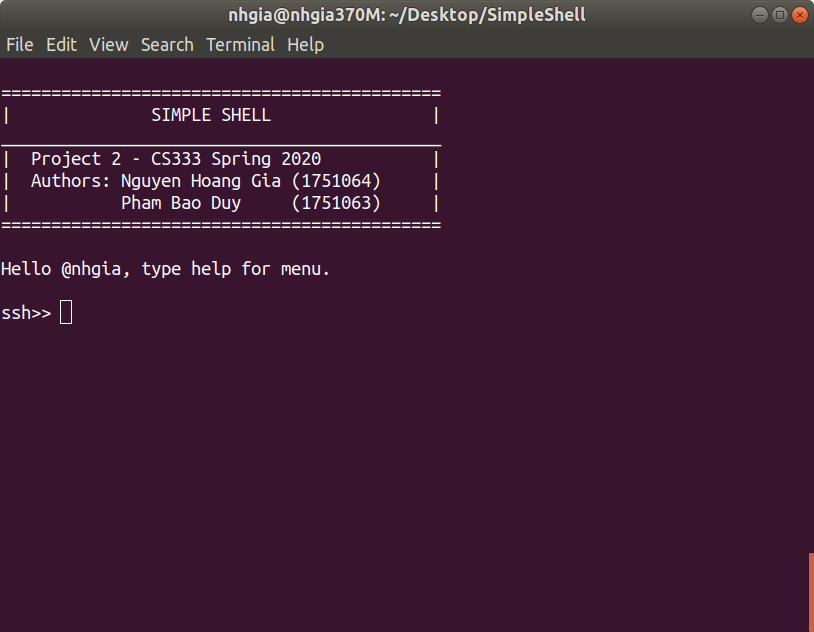
cat < myFile.txt

#### 5. Communication via a Pipe

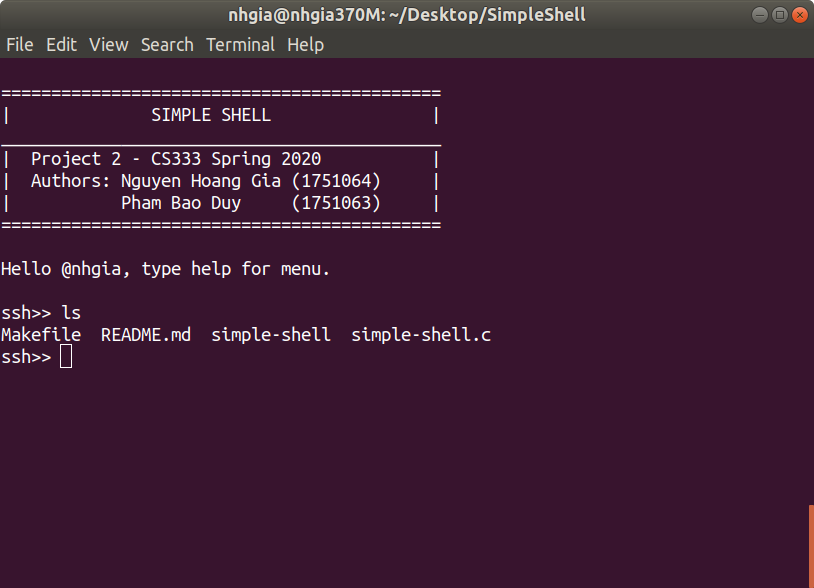
* Output of one command to serve as input to another using a pipe.
* For example: list sub-folder and sub-item in current directory, just display which has character "a" in the name.

ls | grep a

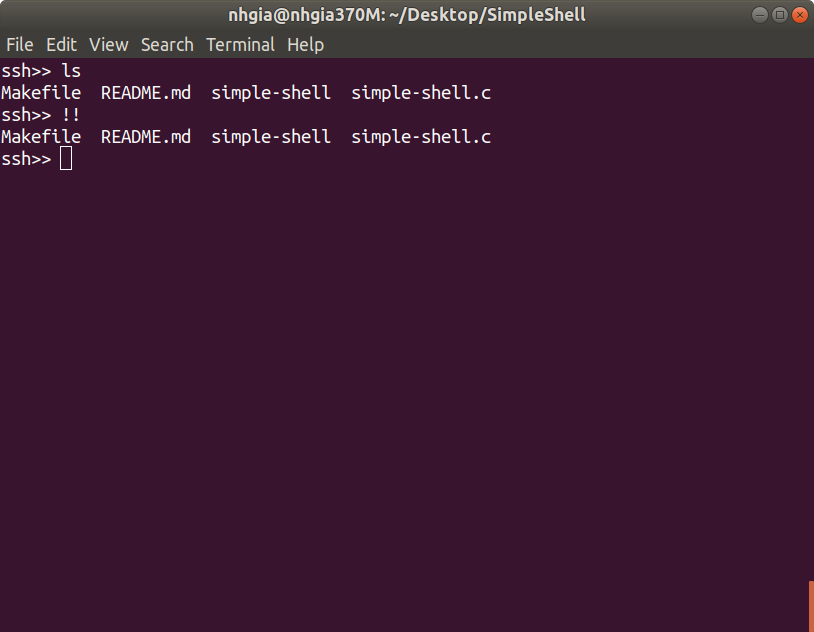
1. **Result report**
2. Main (after running ./simple-shell)



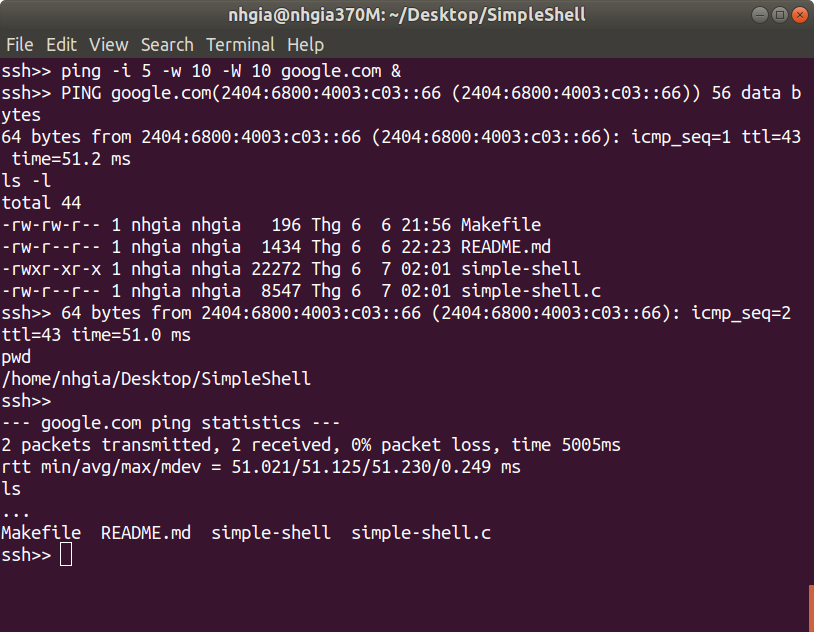
1. Single command with folk() child process



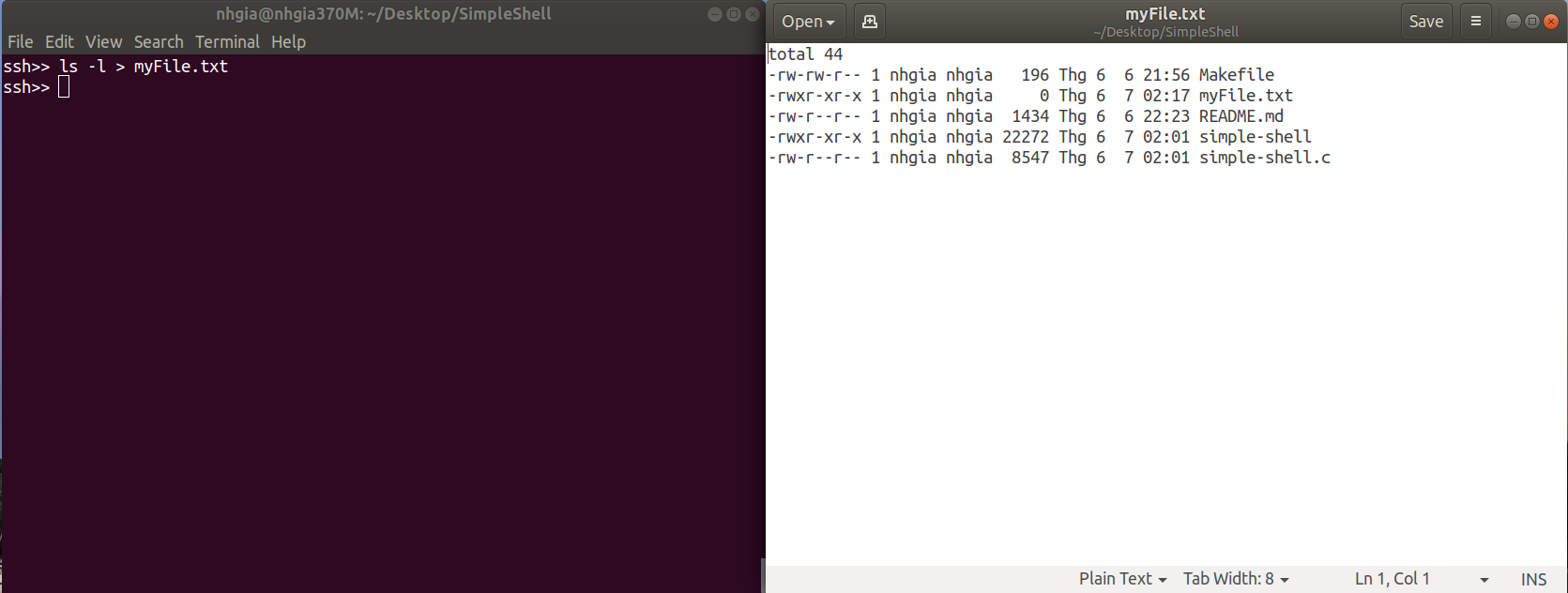
1. History by “!!”



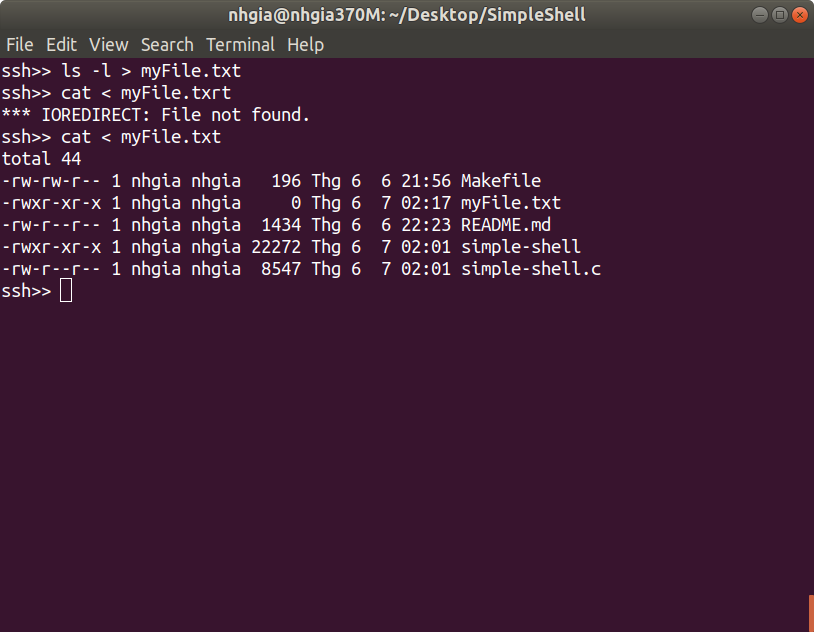
1. Concurrent conmmands



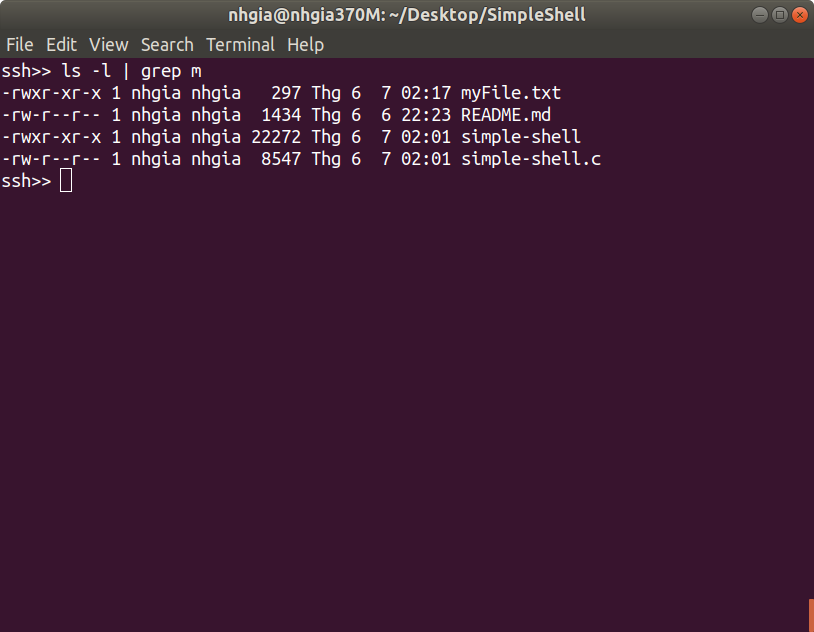
1. Output redirection



1. Input redirection



1. Communication via a Pipe



1. **Code review**

Overall, the program at run time works like normal C program but take input argument from user entered via Terminal.

The main work-flow of simple-shell can be illustrate by this:

Input

Exit

* Pre-process: Flush the input-stream, get new one from fget(), add ‘\0’ for end of user input.
* Next, check whether the input (first argument or args[0]) is special case or not. This includes show history (“!!” command), help menu, exit program, clear screen.
* Function parse() takes an input line and returns a zero-terminated array of char pointers, each of which points to a zero-terminated character string. This function loops until a binary zero is found, which means the end of the input line "command" is reached. If the current character of "command" is not a binary zero, parse() skips all white spaces and replaces them with binary zeros so that a string is effectively terminated. Once parse() finds a non-white space, the address of that location is saved to the current position of "args" and the index is advanced. Then, parse() skips all non-whtitespace characters. This process repeats until the end of string "command" is reached and at that moment "args" is terminated with a zero.
* The execute() function: After get everything from parse, now the argument passed into execute() is an array of character, which element is a part of input. For example: ls -l | grep a will be args[0] = “ls”, args[1] = “-l”, args[2] = “|”, args[3] = “grep”, args[4] = “a”. Next, it continues to check whether it is pipe (contains “|”) or IO redirect (contains “<” or “>”),…
* After got what kind of command from input, we fork child process to execute input command through folk() function. Thanks for the pid\_t data type and also epending on which kind of command, we separate child process logic:

(after fork 1st child process)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| pid\_t | fork() > 0  **At parent process** | fork() == 0  **At child process** | | fork() < 0  **Error** |
| Single command with child process | Wait for child process to be killed. | Execute command. | | Exit() |
| Concurrent | Return to parent process immediately, no need to wait. |
| IO Redirect | Wait for child process to be finished. | * If output: open file, redirection using dup2() function (send output to opened file). * If input: open file, redirection using dup2() function (send file to input). | |
| Pipe | Execute 1st command. | **#1 child** | **#2 child** |
| Fork 2nd child process.  Write to child #2. | Read from child #1.  Execute 2nd command. |
| Wait for both child process to be killed. | |

***- END -***