

My Simple Shell (SiSH)

32190984 김이수

Free days remaining : 5 / Used : 0

Index

1. Building
2. Usage & Features
3. Implementation
4. Limitation & Known Bugs
5. Files
6. ETC

1. Building

In order to use SiSH (My simple shell), you need to download the original source code and build it yourself. This can be achievable in two ways.

- a) Use CMake (requires minimum VERSION 3.22)
- b) Use Makefile

There is CMakelists.txt and Makefile that can automatically build SiSH easily.

```
# For CMake
$ mkdir build && cd build
$ cmake .. && cmake --build .
```

```
# For Makefile
$ make sish
```

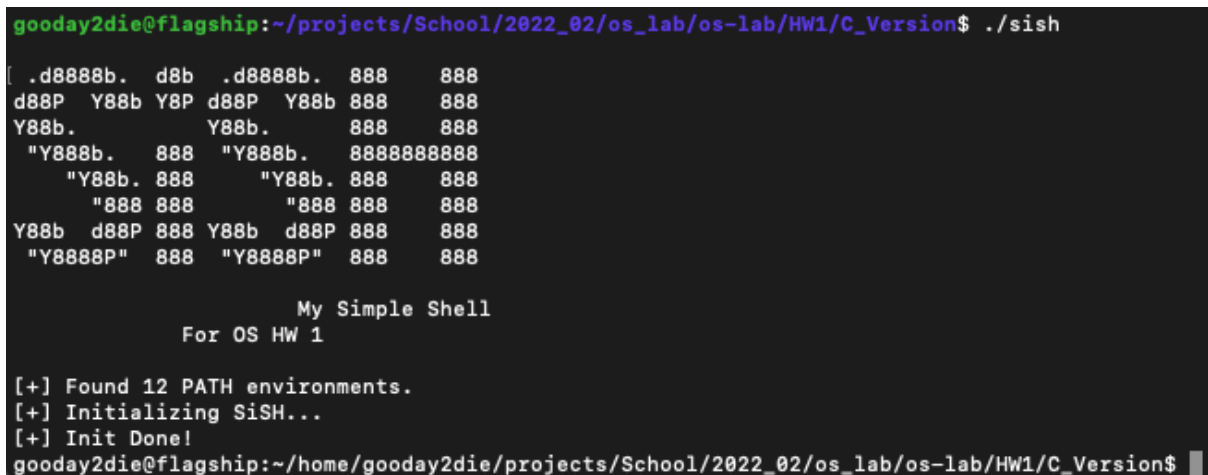
Any of commands above will generate a binary file named “**sish**”.

When using Makefile, Makefile supports two options.

- “make sish” for building SiSH.
- “make clean” for cleaning up every compiled stuff including SiSH.

```
$ ./sish
```

SiSH will welcome you with a small ASCII art. SiSH will initially detect your \$PATH environment settings and store them internally. This might take a while if you have lots of \$PATH environments.



```
gooday2die@flagship:~/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ ./sish

[ .d8888b. d8b .d8888b. 888 888
d88P Y88b Y8P d88P Y88b 888 888
Y88b.      Y88b. 888 888
"Y888b. 888 "Y888b. 88888888888
  "Y88b. 888  "Y88b. 888 888
    "888 888    "888 888 888
Y88b d88P 888 Y88b d88P 888 888
"Y8888P" 888 "Y8888P" 888 888

                My Simple Shell
            For OS HW 1

[+] Found 12 PATH environments.
[+] Initializing SiSH...
[+] Init Done!
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$
```

If you see this screen (might vary depending on your environment), you have now successfully built SiSH and are ready to use SiSH.

2. Usage & Features

You can use SiSH just like bash. (Yes, it lacks some features, but you can pretty much play around with it!) SiSH is currently capable of following operations:

- A) Changing directory.
- B) Execute commands with arguments
- C) Execute local binary files with arguments.
- D) Sending signals.
- E) Dynamic prompt.
- F) Exiting SiSH.

A) Changing Directory

You can change current working directory by command **cd**. You can use relative directory as well as absolute directory. Let's assume that you try "cd ..".

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ cd ..  
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1$ cd ..  
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab$ cd ..  
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab$ cd ..  
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02$
```

When you change directory, the prompt will change as well (This dynamic prompt will be mentioned later). You can also cd into far away directories such as /etc/

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02$ cd /etc  
gooday2die@flagship:~/etc$
```

But for example, if you do not have permission to a specific directory (let's say a directory that you do not own) it will let you know it failed.

```
[gooday2die@flagship:~/etc$ cd /home/isu-test11  
[-] cd failed : Permission denied  
gooday2die@flagship:~/etc$
```

Or if you try to move into a directory that does not exist, it will let you know that the directory does not exist.

```
gooday2die@flagship:~/etc$ cd /unknown  
[-] cd failed : No such file or directory
```

You can freely move around and change directories using cd command.

B) Executing commands

You can execute commands such as **ls** using SiSH. SiSH will automatically find out that your command **ls** is actually **/usr/bin/ls** (for example).

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ ls
CMakeLists.txt  README.md  build  main.o  sish.c  sish.o  utils.c  utils.o
Makefile        a.out     main.c  sish    sish.h  test    utils.h
```

You can use any commands that are registered in your `$PATH` variable directories. Please see limitations for more information. Also you can execute commands with arguments as well! Let's say you would like to execute **ls** with argument **-l** and **-a**. Then, just execute **ls -a -l**. (Of course, you will just use **ls -al** but this is for demonstration)

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ ls -a -l
total 124
drwxrwxr-x 4 gooday2die gooday2die 4096 Sep 21 21:02 .
drwxrwxr-x 4 gooday2die gooday2die 4096 Sep 21 11:23 ..
-rw-rw-r-- 1 gooday2die gooday2die 105 Sep 20 23:51 .gitignore
-rw-rw-r-- 1 gooday2die gooday2die 138 Sep 19 21:21 CMakeLists.txt
-rw-rw-r-- 1 gooday2die gooday2die 282 Sep 19 21:26 Makefile
-rw-rw-r-- 1 gooday2die gooday2die 1596 Sep 14 22:45 README.md
```

Let's assume following scenario.

1. Make directory named a.
2. Move into directory a.
3. Make directory named b.
4. Check current working directory.
5. Rename directory b as c.
6. Execute **ls**

The scenario can be achievable with SiSH just like if you were using bash!

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ mkdir a
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ cd ./a
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version/a$ mkdir b
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version/a$ pwd
/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version/a
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version/a$ mv b c
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version/a$ ls
c
```

Let's assume that you were trying to execute an unknown command.

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version/a$ unknown
[-] Binary or command not found.
```

If SiSH failed to find your command in `$PATH` environment, SiSH will let you know no such command or binary exists.

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ noperm
[-] Execution failed : Permission denied
```

Or, if you were trying to execute a command that you do not have permission to, SiSH will let you know that you do not have permissions. Error messages are not just limited to those two cases. SiSH will automatically generate error messages according to the reason of failing execution.

C) Executing binaries

You can execute local binary files using SiSH with arguments as well. Let's say that we have an binary executable file named **a.out** that prints out "hihihihihih" and prints out all args.

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ ./a.out
[+] Executing ./a.out
hihihihihih
./a.out
```

You can also execute **a.out** with arguments as well. Let's say that you call **a.out** with arguments **foo** and **bar**.

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ ./a.out foo bar
[+] Executing ./a.out
hihihihihih
./a.out
foo
bar
```

Also, if you were trying to execute a non-executable file, SiSH will let you know that the file is non-executable.

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ touch nonEx
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ chmod 777 nonEx
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ ./nonEx
[+] Executing ./nonEx
[-] Execution failed : Exec format error
```

In this case, we made a file named **nonEx** with permission **777** which allows executing the file. However, since the file is not an executable file, SiSH will prompt an error message stating that it failed executing due to **Exec format error**. Let's see another example.

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ sudo touch noPerm
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ sudo chmod 700 noPerm
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ ./noPerm
[+] Executing ./noPerm
[-] Execution failed : Permission denied
```

In this case you have a file named **noPerm** and was owned by **root**. Which means that you do not have permission to execute the file. However, when you try to execute **noPerm** SiSH will fail execution and prompt **Permission denied**. Error messages are not just limited to those two cases. SiSH will automatically generate error messages according to the reason of failing execution.

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ /home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version/a.out foo bar
[+] Executing /home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version/a.out
hihihihihih
/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version/a.out
foo
bar
```

You can also execute your binary file using absolute path as well. With the same **a.out**, the example above will execute your binary file using absolute path with arguments **foo** and **bar**.

D) Sending Signals

You can send signals to processes that SiSH is executing. However, due to technical limitations, the signals that are possible to send is limited to SIGINT. This signal will kill the process that is running.

```
gooday2die@flagship:~/etc$ sleep 5
```

Let's assume that you executed "sleep 5" and would like to kill the process immediately using SIGINT. You can just press **CTRL + C** and that will generate Interrupt signal. This will kill the process immediately.

```
gooday2die@flagship:~/etc$ sleep 5
^C
[+] GOT SIGNAL : Interrupt
```

But, if you just pressed CTRL + C without any process being ran in the background, it will ignore your interrupt signal.

```
gooday2die@flagship:~/etc$ ^C
[+] GOT SIGNAL : Interrupt
^C
[+] GOT SIGNAL : Interrupt
^C
[+] GOT SIGNAL : Interrupt
gooday2die@flagship:~/etc$
```

This will just ignore your signal CTRL + C and keep running the SiSH. (This will not kill SiSH at all!)

E) Dynamic prompt

SiSH provides you a dynamic prompt with following data:

- Username
- Prompt sign
- Your current pwd
- Hostname

When SiSH initializes for the first time, it will gather your hostname and username using function. It will prompt with information.

```
gooday2die@flagship:~/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ ./sish
```

Bash prompts like this (at least in my machine).

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$
```

SiSH prompts like this! Also, when you move around using **cd**, the prompt will change as well.

```
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ cd ..
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1$ cd ..
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab$ cd ..
gooday2die@flagship:~/home/gooday2die/projects/School/2022_02/os_lab$ cd /etc
gooday2die@flagship:~/etc$
```

Also, if you execute with normal user, the prompt will be **\$**. However, if you execute SiSH with **root** permission (which is highly not suggested), the prompt will be **#**.

```
gooday2die@flagship:~/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ sudo ./sish

.d8888b.  d8b  .d8888b.  888    888
d88P  Y88b  Y8P  d88P  Y88b  888    888
Y88b.      Y88b.      888    888
"Y888b.    888  "Y888b.  88888888888
   "Y88b.  888    "Y88b.  888    888
     "888 888      "888 888    888
Y88b  d88P 888  Y88b  d88P 888    888
"Y8888P"  888  "Y8888P"  888    888

                My Simple Shell
                For OS HW 1

Be careful! You are running SiSH as root!
[+] Found 7 PATH environments.
[+] Initializing SiSH...
[+] Init Done!
root@flagship:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version#
```

SiSH will also automatically detect that you are root and warn you to be careful. Remember, with great power comes great responsibility. Also, since SiSH is a toy project, please avoid executing SiSH with root permission as much as possible!

F) Exiting SiSH

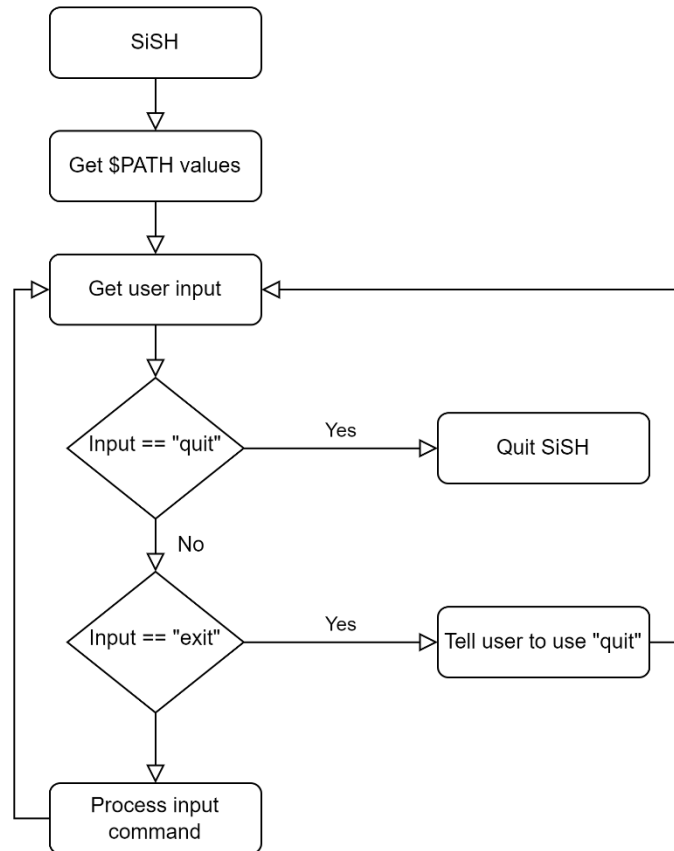
You can quit SiSH with command “quit”. If you execute command “exit” (because of your old habits), SiSH will let you know that you should use “quit” instead of “exit”. With a goodbye message.

```
gooday2die@unknown:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ exit
[-] Use quit instead
gooday2die@unknown:~/home/gooday2die/projects/School/2022_02/os_lab/os-lab/HW1/C_Version$ quit

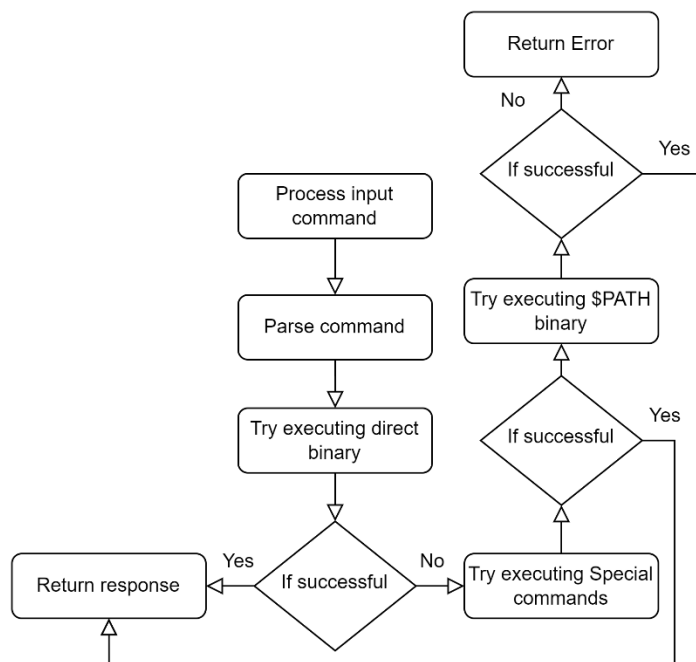
.d8888b.      888      888
d88P  Y88b      888      888
888   888      888      888
888           .d88b.   .d88b.   .d888888  888888b.  888  888  .d88b.
888  888888  d88" "88b  d88" "88b  d88"  888      888  "88b  888  888  d8P  Y8b
888   888  888  888  888  888  888  888  888  888888888  888  888  888  888888888
Y88b  d88P  Y88..88P  Y88..88P  Y88b  888      888  d88P  Y88b  888  Y8b.
"Y8888P88  "Y88P"   "Y88P"   "Y88888  888888P"  "Y88888  "Y8888
                                     888
                                     Y8b  d88P
                                     "Y88P"
```

3. Implementation

Implementation and lifecycle of SiSH can be represented as below with flowchart:



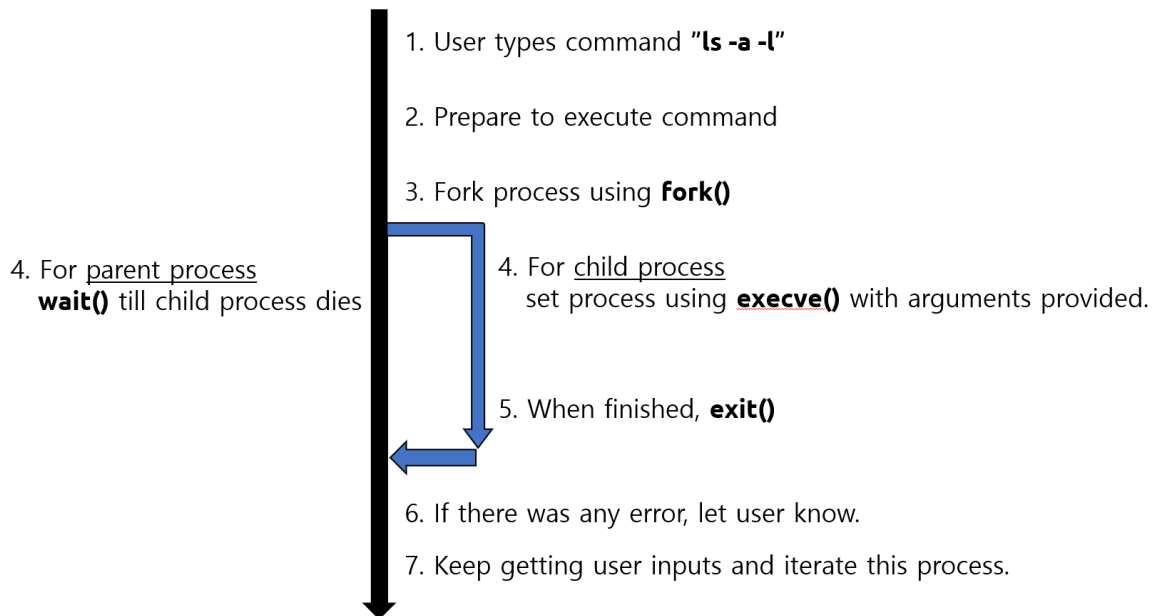
For processing command, the following procedure happens:



For executing commands, SiSH does following tasks

1. Fork current process using **fork()**.
2. For parent process, tell it to wait until child process dies using **wait()**
3. For child process, tell it to **execve()** with arguments provided.

For example, let's say user issued command "**ls -a -l**". Following procedure will happen.



In short, SiSH uses **fork()** to generate a new process. Then it will set the child process using **execve()** with the arguments parsed. The commands will be parsed into an array of **char*** and will be fed into **execve()** as arguments with NULL termination.

4. Limitation & Known Bugs

SiSH is not bash. SiSH has some limitations and known bugs. Those are listed below.

- a) Max length of string that represents each PATH variable is 100. (Meaning that if a PATH variable is set more than 100 characters, unexpected behavior might happen). If you wish to make your PATH variable limit go beyond 100 characters, modify "BUFFER_SIZE" from "utils.h".
- b) For binaries that are executed by PATH variables, SiSH will execute the fastest binary that was found.

Let's say that you have binary named "python" in both "/usr/bin/" and "/usr/local/bin". Also, with "echo \$PATH", the order of PATH variable is like "/usr/bin/:.....:/usr/local/bin". In this case, SiSH will execute "/usr/bin/python" instead of "/usr/local/bin/python" since it had lower index and was found earlier.

- c) For command inputs, the maximum length that a command can be is 300 characters. If you go beyond that limit, the command might not work properly or even affect the command that is coming after this current command. If you wish to modify your maximum limit of command, please modify "MAX_COMMAND_LEN" from "sish.h".
- d) For dynamic prompt that shows current working directory on prompt, SiSH can only represent 300 characters. If the current working directory has more than 300 characters, unexpected behavior might occur. If you wish to modify your maximum limit of command, please modify "MAX_CWD_LEN" from "sish.h".
- e) Command **clear** will print out 100 new lines. Due to technical limitation, SiSH is not registered as unknown X-term. In order for original command **clear** to work, it needs X-term information. To solve this problem, SiSH will just print 100 new lines to pretend that the command **clear** was processed. The count of new lines can be changed via "CLEAR_NEW_LINES" in "utils.h".
- f) SiSH uses **gethostname** to find hostname for dynamic prompt. If getting hostname failed, unexpected behavior might happen.
- g) For signal SIGTSTP which can be fired by **CTRL + Z**, SiSH might freeze due to error. Only SIGINT which can be fired by **CTRL + C** can be processed properly.
- h) SiSH uses **fgets()** in order to get user's input. Do not use **CTRL + D** which is **EOF**. This will mess with SiSH and will break SiSH.
- i) Warning: Avoid using **root** permission for executing SiSH. Remember, with great power comes great responsibility. SiSH is not tested enough to say using **root** permission with SiSH is "safe". Remember, this is a toy project.

5. Files

SiSH comes with following files:

- CMakeLists.txt
- Makefile
- main.c
- sish.c & sish.h
- utils.c & utils.h

CMakeLists.txt

CMakeLists.txt exist for building SiSH with CMake if you would like to. Requires minimum VERSION 3.22.

Makefile

Makefile includes recipes for building SiSH and cleaning up the space. Currently, Makefile supports two recipes:

- **make sish**: For building SiSH.
- **make clean**: For cleaning every *.o files and removing SiSH executable file.

main.c

main.c has main function for entry point. There is a hidden option named “**iamdev**” which enables developer mode. This will show you all directories registered in \$PATH variable.

sish.c & sish.h

- **sish.c**: Contains codes that implements functions that were declared by sish.h.
- **sish.h**: Contains codes that declare functions that are required to run SiSH. These include:
 - Processing commands (**processCommand**).
 - Executing with **fork()**, **execve()** and **wait()** (**execute**).
 - Finding binary executable file in all directories registered in \$PATH variable (**findWhichBin**).
 - Processing local binary executable file (**processDirectBinary**).
 - Processing special commands such as **clear** and **cd** (**processSpecialCommand**).
 - Processing binary executable file in \$PATH directory (**executePathBinary**).
 - Printing errors (**printError**).
 - Handling signals (**alarm_handler** and **child_handler**)

utils.c & utils.h

- **utils.c**: Contains codes that implements functions that were declared by sish.h.
- **utils.h**: Contains codes that declare functions that are required for utilities. These include:
 - Getting directories registered in \$PATH (**getPaths**).
 - Parsing commands with arguments (**parseCommand**).
 - Printing banner and goodbye message (**printBanner**, **printBye**).

6. ETC

For SiSH following programming conventions were used:

- K&R style and Clang-tidy
- Docstring format function documentation

SiSH uses C99 and was meant to be built in POSIX environment. SiSH is OS dependent. Thus, MinGW or Visual Studio compiler will fail compiling SiSH.

SiSH was tested working with following environments

- Personal server: Ubuntu 22.04
- Assam server (assam.dankook.ac.kr): Ubuntu 16.04