CShell

Luca Colarossi

31. Juli 2023

Inhaltsverzeichnis

CShell is a terminal that allows you to perform certain actions with certain commands. It includes the standard commands, a module system and a server that allows you to connect remotely from another computer using the proprietary NSSH protocol.

1 installation

To install CShell, run the command "**git clone https://github.com/Innocent213/CShell**ïn the terminal. After that, go to the **CShell** folder and run the command "**sudo chmod 777 Compile.sh**". Next run the command "**sudo bash Compile.sh**". Then run the command "**bash StartCShell.sh**änd follow the instructions.

2 commands

2.1 touch

The 'touch' command creates a file and changes the access date of an existing file.
SYNTAX: **touch** [**FILEPATH**]

2.2 ls

The 'ls' command lists all files within a folder. SYNTAX: **ls** [**OPTIONS**]
OPTIONS:

- -a | Displays all files, including hidden ones.
- -l | Show all files with their permissions

2.3 cat

The command 'ls' lists all files inside a folder. SYNTAX: **cat** [**FILEPATH**]

2.4 clear

The 'clear' command clears the console Ṡ YNTAX: **cat**

2.5 mkdir

The 'mkdir' command creates a folder in a specified directory.

2.6 chmod

The 'chmod' command changes the permissions of a file or folder. SYNTAX: **ls** [**OPTIONS**] [**FILEPATH**]

2.7 man

The 'man' command displays the man page of a command.
SYNTAX: **man** [**COMMAND**]

2.8 cd

The command 'cd' changes the working directory. SYNTAX: **cd** [**FOLDERPATH**]

2.9 cwd

The command 'cwd' changes the working directory

2.10 help

The 'help' command prints a list of available commands.

2.11 rm

The command 'rm' deletes a file or folder (recursively) SYNTAX: **rm** [**OPTIONS**] [**FILE-PATH**]
OPTIONS:

- -r | Delete a folder recursively

2.12 module

The 'module' command manages the modules.ṠYNTAX: **module** [**OPTIONS**]
OPTIONS:

- list | Lists all modules (color red: switched off | color green: switched on)
- reload | Reload all modules

or SYNTAX: **module** [**MODULE**] [**OPTIONS**]
OPTIONS:

- info | Reveals information about the module(description, auto, version, name)
- disable | Turns the module off
- enable | Turns the module on

2.13 libary

The 'libary' command manages the libraries for the modules. SYNTAX: **module** [**OPTIONS**]
OPTIONS:

- list | Lists all libraries in the database
- remove | Removes a libary –> libary remove [LIBARY_NAME]
- add | Remove a library –> libary add [LIBARY_FOLDER] [LIBARY_NAME]

2.14 nssh

The 'module' command manages the modules.ṠYNTAX: **module** [**OPTIONS**]
OPTIONS:

- clients | Lists all connected and locked devices with their names.
- port | 1 argument: prints the currently configured port. 2 arguments: store a new port -> nssh port [PORT]
- enable | Enable the NSSH server
- disable | Disables the NSSH server
- state | Shows the status of the NSSH server

or SYNTAX: **nssh** [**CLIENT**] [**OPTIONS**]
OPTIONS:

- ban | Disables a connected device.
- unban | Unlocks a locked device.
- kick | Terminate the connection of a device.

3 Modules

Modules are the best way to integrate your own C code into the CShell. For this I have programmed a C library, which makes this very easy. It is called CSHLib and is located in the **lib** folder in the same folder where the CShell is located. Now to program a module you have to add the header file **CSHLib.h** to the code:

```c
#include "../lib/CSHLib.h"

int main(int argc, char *argv[]) {
    return 0;
}
```

Next, the initialization code is built in. This consists of two parts. The first is the function **csh_InitModule**. This is passed the argument variables *argv* and *argc* from the main function:

```c
#include "../lib/CSHLib.h"

int main(int argc, char *argv[]) {
    csh_InitModule(argv, argc);
    return 0;
}
```

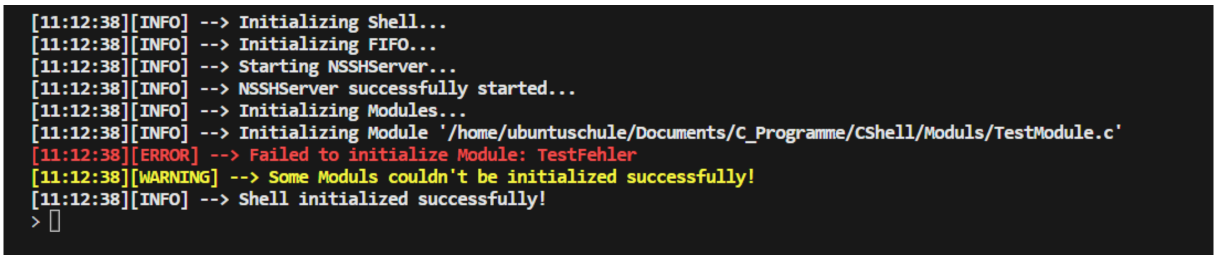
The second part is **csh_BuildModule**. Here the following parameters are asked:

- **n**ame | name of the module
- **A**uthor | Author of the module
- **V**ersion | Version of the module
- **D**escription | Description of the module
- **I**nizialize function | A function that will be called when the module is compiled and loaded.

```c
#include "../lib/CSHLib.h"

char *onInit() {
    return NULL;
}

int main(int argc, char *argv[]) {
    csh_InitModule(argv, argc);
    csh_BuildModule("TestModule", "Author", "1.0", "This is a test module", onInit)
    ;
    return 0;
}
```

If the onInit function returns NULL as a pointer, it means to the CShell that no error occurred during the process. If a string (char array) is returned, it means for the module and an error will be issued:

```c
#include "../lib/CSHLib.h"

char *onInit() {
    return "TestError";
}

int main(int argc, char *argv[]) {
    csh_InitModule(argv, argc);
    csh_BuildModule("TestModule", "Author", "1.0", "This is a test module", onInit)
    ;
    return 0;
}
```



```
[11:12:38][INFO] --> Initializing Shell...
[11:12:38][INFO] --> Initializing FIFO...
[11:12:38][INFO] --> Starting NSSHServer...
[11:12:38][INFO] --> NSSHServer successfully started...
[11:12:38][INFO] --> Initializing Modules...
[11:12:38][INFO] --> Initializing Module '/home/ubuntuschule/Documents/C_Programme/CShell/Moduls/TestModule.c'
[11:12:38][ERROR] --> Failed to initialize Module: TestFehler
[11:12:38][WARNING] --> Some Moduls couldn't be initialized successfully!
[11:12:38][INFO] --> Shell initialized successfully!
> 
```

Abbildung 1: example CShell

3.1 commands

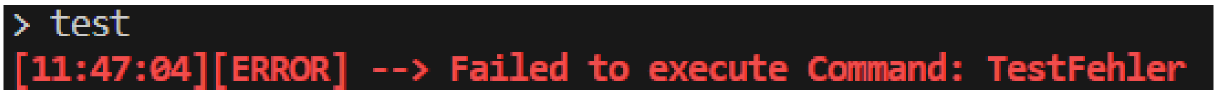
In order to implement commands into the module, you have to regeminate it. This happens between initializing and building the module. For this you use the method **csh_RegisterCommand**. It takes the following parameters:

- **C**ommand | The command to execute in the CShell
- **A**liases | Other spellings of the command
- **A**lias Count | The number of 'aliases'.
- **D**escription | Short description of what the command does
- **M**anpage | An optional manpage (see next subsection)
- **E**xecution Function | A function that will be executed when the **command** or one of the **alias** is executed in the CShell.

```c
#include "../lib/CSHLib.h"

char *onTestExecution(char *cmd, char *args[], int args_count) {
    return NULL;
}

char *onInit() {
    return NULL;
}

int main(int argc, char *argv[]) {
    csh_InitModule(argv, argc);

    char *test_aliases[10] = { "te" };
    csh_RegisterCommand("test", test_aliases, 1, "This is a test command.", NULL,
    onTestExecution);

    csh_BuildModule("TestModule", "Author", "1.0", "This is a test module", onInit)
    ;
    return 0;
}
```

If the onExecution function returns NULL as a pointer, it means to the CShell that no error occurred during the process. If a string (char array) is returned, it means for the module and an error will be issued:

```c
#include "../lib/CSHLib.h"

char *onTestExecution(char *cmd, char *args[], int args_count) {
    return "TestError"
}

char *onInit() {
    return NULL;
}

int main(int argc, char *argv[]) {
    csh_InitModule(argv, argc);

    char *test_aliases[10] = { "te" };
    csh_RegisterCommand("test", test_aliases, 1, "This is a test command.", NULL,
    onTestExecution);

    csh_BuildModule("TestModule", "Author", "1.0", "This is a test module", onInit)
    ;
    return 0;
}
```



```
> test
[11:47:04][ERROR] --> Failed to execute Command: TestFehler
```

Abbildung 2: example CShell

3.2 manpages

Manpages allow you to document the function of a command in detail, using formatting as well. A man page consists of a title and paragraphs. For example, a man page can be created like this;

```c
#include "../lib/CSHLib.h"

char *onTestExecution(char *cmd, char *args[], int args_count) {
    return NULL;
}

char *onInit() {
    return NULL;
}

int main(int argc, char *argv[]) {
    csh_InitModule(argv, argc);

    CSHManpage *manpage = Manpage_Init();
    ManpageParagraph *paragraph = Manpage_CreateParagraph();
    ManpageTitle *title = Manpage_CreateTitle("TestManpage");
    ManpageTitle *paragraph_title = Manpage_CreateTitle("TestParagraph");
    ManpageContent *paragraph_content = Manpage_CreateContent("This is a test
    command!");

    Manpage_SetTitleFormat(title, COLOR_RED, true, true, false);
    Manpage_SetTitleFormat(title, COLOR_CYAN, false, false, false);
    Manpage_SetContentFormat(paragraph_content, COLOR_MAGENTA, false, false, false)
    ;

    Manpage_SetParagraphTitle(paragraph, paragraph_title);
    Manpage_SetParagraphContent(paragraph, paragraph_content);
    Manpage_AddParagraph(manpage, paragraph);
    Manpage_SetTitle(manpage, title);

    char *test_aliases[10] = { "te" };
    csh_RegisterCommand("test", test_aliases, 1, "This is a test command.", NULL,
    onTestExecution);

    csh_BuildModule("TestModule", "Author", "1.0", "This is a test module", onInit)
    ;
    return 0;
}
```

With the function **Manpage__Init** the manpage is inizialized. Next the **Manpage__CreateParagraph** function creates a new paragraph. After that the **Manpage__CreateTitle** function is used to create a title for the man page. function to create a title for the man page with the text ***TestManpage*** and one for the paragraph with the text ***TestParagraph***. The **Manpage__CreateContent** function creates the content for the previously created paragraph with the text ***This is a test command!***. The **Manpage__SetTitleFormat** and **Manpage__SetContentFormat** functions set the format for the paragraph title and content and the title for the man page. By **Manpage__SetParagraphTitle**, **Manpage__SetParagraphContent**, **Manpage__AddParagraph**, **Manpage__SetTitle** functions the manpage is assembled and then it is passed as 5th parameter into **csh__RegisterCommand** function.

3.3 Events

Another function of the CShell are events. These work in such a way that a module can trigger an event and another module or its own can receive it and execute code. With the execution of an event also JSON data can be transferred, which can then be processed by the modules, which have registered the event. You can regest an event with the **csh__RegisterEvent** regest:

```c
#include "../lib/CSHLib.h"

```

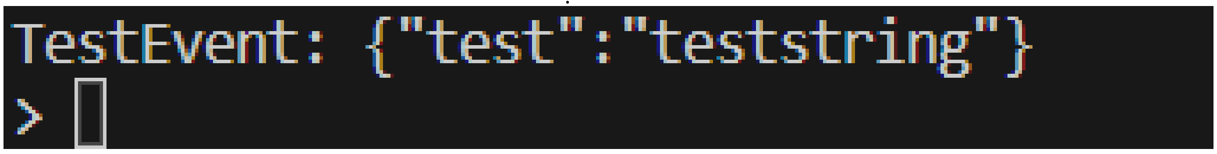
```c
char *onTestEvent(cJSON *data) {
    csh_printNormal("TestEvent: ");
    csh_printJSON(data);
    csh_printNormal("\n");
    return NULL;
}

char *onInit() {
    return NULL;
}

int main(int argc, char *argv[]) {
    csh_InitModule(argv, argc);

    csh_RegisterEvent("testevent", onTestEvent);

    csh_BuildModule("TestModule", "Author", "1.0", "This is a test module", onInit)
    ;
    return 0;
}
```

To trigger an event, use the function **csh_callEvent**. 2 parameters are passed:

- **name** | The name of the event to trigger.
- **Data** | The JSON data to be transferred when the event is triggered.

```c

    csh_callEvent("testevent", data);
```

.



```
TestEvent: {"test":"teststring"}
> 
```

Abbildung 3: example CShell

3.4 actions

Functions that can control the CShell:

- csh_printNormal | Prints a normal text.
- csh_printInfo | Print a text as information.
- csh_printError | Output a text as error.
- csh_printFatal | Output a text as fatal error.
- csh_printJSON | Print a JSON object as text.
- csh_Shutdown | Turns off the CShell.
- csh_ExecuteCommand | Executes a command on the CShell -> Returns an error code.

```c

    void csh_printNormal(const char *format, ...);
    void csh_printInfo(const char *format, ...);
    void csh_printError(const char *format, ...);
    void csh_printFatal(const char *format, ...);
    void csh_printJSON(cJSON *json);

    void csh_Shutdown(int error_code);
    int csh_ExecuteCommand(char *str, bool silent_mode);
```

4 NSSH protocol

I have programmed the NSSH protocol(Network Secure Shell) especially for the CShell. This protocol uses **JSON**(Javascript Object Notation) and **SSL/TSL** (Secure Sockets Layer/Transport Layer Security) to communicate and interact with the CShell. You can find the client for this in the **Utils** folder, in the same folder where the CShell is located. When running it, you just need to enter the IP and port (default: 8921). If you get the error **Cannot connect to Server: You are banned!**, it means that your PC has been banned from the CShell. If the error message **Cannot connect to Server: Unknown Reason!**, the client cannot connect to the client for unknown reasons.