

# CShell

Luca Colarossi

9. Juni 2023

## Inhaltsverzeichnis

<b>Inhaltsverzeichnis</b>	<b>1</b>
<b>1 Installation</b>	<b>2</b>
<b>2 Befehle</b>	<b>2</b>
2.1 touch . . . . .	2
2.2 ls . . . . .	2
2.3 cat . . . . .	2
2.4 clear . . . . .	2
2.5 mkdir . . . . .	2
2.6 chmod . . . . .	2
2.7 man . . . . .	2
2.8 cd . . . . .	3
2.9 cwd . . . . .	3
2.10 help . . . . .	3
2.11 rm . . . . .	3
2.12 module . . . . .	3
2.13 libary . . . . .	3
2.14 nssh . . . . .	4
<b>3 Module</b>	<b>4</b>
3.1 Kommandos . . . . .	5
3.2 Manpages . . . . .	7
3.3 Ereignisse . . . . .	7
3.4 Aktionen . . . . .	8
<b>4 NSSH-Protokoll</b>	<b>9</b>

Die CShell ist ein Terminal das es Ihnen erlaubt mit bestimmten Befehlen bestimmte Aktionen auszuführen. Es beinhaltet die Standardbefehle, ein Modulsystem und einen Server womit man mithilfe des Hauseigenen NSSH-Protokolls eine Remoteverbindung von einem anderen Computer erlaubt.

## 1 Installation

Um die CShell zu installieren, führe den Befehl "**git clone https://github.com/Innocent213/CShell**" im Terminal aus. Danach führe du in den Ordner **CShell** und führe den Befehl "**sudo chmod 777 Compile.sh**". Als nächstes führe den Befehl "**sudo bash Compile.sh**" aus. Danach führe den Befehl "**bash StartCShell.sh**" aus und folge den Anweisungen.

## 2 Befehle

### 2.1 touch

Der Befehl 'touch' erstellt eine Datei und ändert das Zugriffsdatum einer bereits bestehenden Datei.

SYNTAX: **touch** [ **FILEPATH** ]

### 2.2 ls

Der Befehl 'ls' listet alle Dateien innerhalb eines Ordners auf.

SYNTAX: **ls** [ **OPTIONS** ]

OPTIONS:

- -a | Zeigt alle Dateien an, auch die versteckten.
- -l | Zeigt alle Dateien mit den dazugehörigen Berechtigungen an

### 2.3 cat

Der Befehl 'cat' listet alle Dateien innerhalb eines Ordners auf.

SYNTAX: **cat** [ **FILEPATH** ]

### 2.4 clear

Der Befehl 'clear' leert die Konsole

SYNTAX: **cat**

### 2.5 mkdir

Der Befehl 'mkdir' erstellt einen Ordner in einem bestimmten Verzeichnis.

### 2.6 chmod

Der Befehl 'chmod' ändert die Berechtigungen einer Datei oder eines Ordners.

SYNTAX: **ls** [ **OPTIONS** ] [ **FILEPATH** ]

### 2.7 man

Der Befehl 'man' zeigt die Manpage eines Befehls an.

SYNTAX: **man** [ **COMMAND** ]

## 2.8 cd

Der Befehl 'cd' ändert das Arbeitsverzeichnis (Working Directory)

SYNTAX: **cd** [ **FOLDERPATH** ]

## 2.9 cwd

Der Befehl 'cwd' ändert das Arbeitsverzeichnis (Working Directory)

## 2.10 help

Der Befehl 'help' gibt eine Liste der verfügbaren Befehle aus.

## 2.11 rm

Der Befehl 'rm' löscht eine Datei oder einen Ordner (rekursiv)

SYNTAX: **rm** [ **OPTIONS** ] [ **FILEPATH** ]

OPTIONS:

- -r | Löscht einen Ordner rekursiv

## 2.12 module

Der Befehl 'module' verwaltet die Module.

SYNTAX: **module** [ **OPTIONS** ]

OPTIONS:

- list | Listet alle Module auf (Farbe Rot: Ausgeschaltet | Farbe Grün: Angeschaltet)
- reload | Ladet alle Module neu

oder

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

OPTIONS:

- info | Gibt Informationen über das Modul preis(Beschreibung, Auto, Version, Name)
- disable | Schaltet das Moduls aus
- enable | Schaltet das Moduls ein

## 2.13 library

Der Befehl 'library' verwaltet die Bibliotheken für die Module.

SYNTAX: **module** [ **OPTIONS** ]

OPTIONS:

- list | Listet alle Bibliotheken in der Datenbank auf
- remove | Entfernt ein Library -> library remove [ **LIBRARY\_NAME** ]
- add | Entfernt ein Library -> library add [ **LIBRARY\_FOLDER** ] [ **LIBRARY\_NAME** ]

## 2.14 nssh

Der Befehl 'module' verwaltet die Module.

SYNTAX: **module** [ **OPTIONS** ]

OPTIONS:

- **clients** | Listet alle verbundenen und gesperrten Geräte mit ihren Namen auf
- **port** | 1 Argument: Gibt den aktuell konfigurierten Port aus. 2 Argumente: speichert einen neuen Port ab -> **nssh port** [ **PORT** ]
- **enable** | Aktiviert den NSSH-Server
- **disable** | Deaktiviert den NSSH-Server
- **state** | Zeigt den Status des NSSH-Servers

oder

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

OPTIONS:

- **ban** | Sperrt ein verbundenes Gerät.
- **unban** | Entsperrt ein gesperrtes Gerät.
- **kick** | Beendet die Verbindung eines Gerätes.

## 3 Module

Module sind der beste Weg eigenen C Code in die CShell zu integrieren. Dafür habe ich eine C Bibliothek programmiert, womit dies ganz einfach möglich ist. Sie heißt CSHLib und befindet sich im **lib** Ordner im gleichem Ordner, in dem sich die CShell befindet. Um nun ein Modul zu programmieren, musst du die Header Datei **CSHLib.h** in den Code einfügen:

```
1 #include "../lib/CSHLib.h"
2
3 int main(int argc, char *argv[]) {
4     return 0;
5 }
```

Als nächstes wird der Initialisierungscode eingebaut. Dieser besteht aus zwei Teilen. Der erste ist die Funktion **csh\_InitModule**. Dieser werden die Argumentvariablen *argv* und *argc* aus der Main Funktion übergeben:

```
1 #include "../lib/CSHLib.h"
2
3 int main(int argc, char *argv[]) {
4     csh_InitModule(argv, argc);
5     return 0;
6 }
```

Der zweite Teil ist **csh\_BuildModule**. Hier werden folgende Parameter gefragt:

- **Name** | Name des Moduls
- **Author** | Autor des Moduls
- **Version** | Version des Moduls
- **Description** | Beschreibung des Moduls
- **Initialize Funktion** | Eine Funktion, die aufgerufen wird, sobald das Modul kompiliert und geladen wird.

```

1 #include "../lib/CSHLib.h"
2
3 char *onInit() {
4     return NULL;
5 }
6
7 int main(int argc, char *argv[]) {
8     csh_InitModule(argv, argc);
9     csh_BuildModule("TestModul", "Author", "1.0", "Das ist ein Test Modul", onInit)
10    ;
11    return 0;
12 }

```

Wenn die onInit Funktion NULL als Pointer zurückgibt, bedeutet das für die CShell, dass kein Fehler während des Prozesses aufgetreten ist. Wenn ein String (Char Array) zurückgegeben wird, bedeutet das für das Modul und es wird ein Fehler ausgegeben:

```

1 #include "../lib/CSHLib.h"
2
3 char *onInit() {
4     return "TestFehler";
5 }
6
7 int main(int argc, char *argv[]) {
8     csh_InitModule(argv, argc);
9     csh_BuildModule("TestModul", "Author", "1.0", "Das ist ein Test Modul", onInit)
10    ;
11    return 0;
12 }

```

```

[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/ubuntu/schule/Documents/C_Programme/CShell/Moduls/TestModule.c'
[11:12:38][ERROR] --> Failed to initialize Module: TestFehler
[11:12:38][WARNING] --> Some Modules couldn't be initialized successfully!
[11:12:38][INFO] --> Shell initialized successfully!
> 

```

Abbildung 1: Beispiel CShell

### 3.1 Kommandos

Um Kommandos in das Modul zu implementieren, muss man ihn registrieren. Das passiert zwischen dem initialisieren und bauen des Moduls. Dazu benutzt du die Methode `csh_RegisterCommand`. Diese nimmt folgende Parameter an:

- **Command** | Der Befehl, der in der CShell ausgeführt werden muss
- **Aliases** | Andere Schreibweisen des Befehls
- **Alias Count** | Die Anzahl der 'Aliases'
- **Description** | Kurze Beschreibung, was der Befehl macht
- **Manpage** | Eine optionale Manpage (siehe nächstes Unterkapitel)
- **Execution Function** | Eine Funktion, die ausgeführt wird, sobald in der CShell der **Command** oder eine der **Aliases** ausgeführt wird.

```

1 #include "../lib/CSHLib.h"
2
3 char *onTestExecution(char *cmd, char *args[], int args_count) {
4     return NULL;
5 }
6
7 char *onInit() {
8     return NULL;
9 }
10
11 int main(int argc, char *argv[]) {
12     csh_InitModule(argv, argc);
13
14     char *test_aliases[10] = { "te" };
15     csh_RegisterCommand("test", test_aliases, 1, "Das ist ein Testbefehl.", NULL,
16         onTestExecution);
17
18     csh_BuildModule("TestModul", "Author", "1.0", "Das ist ein Test Modul", onInit)
19     ;
20     return 0;
21 }

```

Wenn die onExecution Funktion NULL als Pointer zurückgibt, bedeutet das für die CShell, dass kein Fehler während des Prozesses aufgetreten ist. Wenn ein String (Char Array) zurückgegeben wird, bedeutet das für das Modul und es wird ein Fehler ausgegeben:

```

1 #include "../lib/CSHLib.h"
2
3 char *onTestExecution(char *cmd, char *args[], int args_count) {
4     return "TestFehler";
5 }
6
7 char *onInit() {
8     return NULL;
9 }
10
11 int main(int argc, char *argv[]) {
12     csh_InitModule(argv, argc);
13
14     char *test_aliases[10] = { "te" };
15     csh_RegisterCommand("test", test_aliases, 1, "Das ist ein Testbefehl.", NULL,
16         onTestExecution);
17
18     csh_BuildModule("TestModul", "Author", "1.0", "Das ist ein Test Modul", onInit)
19     ;
20     return 0;
21 }

```

```

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

```

Abbildung 2: Beispiel CShell

## 3.2 Manpages

Mit Manpages kannst du die Funktion eines Befehls ausführlich dokumentieren und dabei auch Formatierungen verwenden. Eine Manpage besteht aus einem Titel und Paragraphen. Eine Manpage kann z.B. so erstellt werden;

```
1 #include "../lib/CSHLib.h"
2
3 char *onTestExecution(char *cmd, char *args[], int args_count) {
4     return NULL;
5 }
6
7 char *onInit() {
8     return NULL;
9 }
10
11 int main(int argc, char *argv[]) {
12     csh_InitModule(argv, argc);
13
14     CSHManpage *manpage = Manpage_Init();
15     ManpageParagraph *paragraph = Manpage_CreateParagraph();
16     ManpageTitle *title = Manpage_CreateTitle("TestManpage");
17     ManpageTitle *paragraph_title = Manpage_CreateTitle("TestParagraph");
18     ManpageContent *paragraph_content = Manpage_CreateContent("Das ist ein Test
19     Befehl!");
20
21     Manpage_SetTitleFormat(title, COLOR_RED, true, true, true, false);
22     Manpage_SetTitleFormat(title, COLOR_CYAN, false, false, false, false);
23     Manpage_SetContentFormat(paragraph_content, COLOR_MAGENTA, false, false, false,
24     false);
25
26     Manpage_SetParagraphTitle(paragraph, paragraph_title);
27     Manpage_SetParagraphContent(paragraph, paragraph_content);
28     Manpage_AddParagraph(manpage, paragraph);
29     Manpage_SetTitle(manpage, title);
30
31     char *test_aliases[10] = { "te" };
32     csh_RegisterCommand("test", test_aliases, 1, "Das ist ein Testbefehl.", NULL,
33     onTestExecution);
34
35     csh_BuildModule("TestModul", "Author", "1.0", "Das ist ein Test Modul", onInit);
36
37     return 0;
38 }
```

Mit der Funktion **Manpage\_Init** wird die Manpage initialisiert. Als nächstes wird mit der **Manpage\_CreateParagraph** ein neuer Paragraph erstellt. Danach wird noch mit der Funktion **Manpage\_CreateTitle** Funktion ein Titel für die Manpage mit dem Text *TestManpage* und einen für den Paragraphen mit dem Text *TestParagraph* erstellt. Mit der Funktion **Manpage\_CreateContent** wird der Inhalt für den vorher erstellten Paragraphen mit dem Text *Das ist ein Test Befehl!* erzeugt. Durch die Funktionen **Manpage\_SetTitleFormat** und **Manpage\_SetContentFormat** wird das Format für die Titel und den Inhalt des Paragraphen und der Titel für die Manpage gesetzt. Durch die Funktionen **Manpage\_SetParagraphTitle**, **Manpage\_SetParagraphContent**, **Manpage\_AddParagraph**, **Manpage\_SetTitle** wird die Manpage zusammengefügt und danach wird es als 5. Parameter in die **csh\_RegisterCommand** Funktion übergeben.

## 3.3 Ereignisse

Eine weitere Funktion der CShell sind Ereignisse (Events). Diese funktionieren so, dass ein Modul ein Ereignis auslöst kann und ein anderes Modul oder das eigene, kann es empfangen und Code ausführen. Mit dem ausführen eines Ereignisses können auch JSON Daten übertragen werden, die

dann die Module, die das Event registriert haben weiter verarbeiten können. Ein Event registriert man mit der `csch_RegisterEvent` registrieren:

```

1 #include "../lib/CSHLib.h"
2
3 char *onTestEvent(cJSON *data) {
4     csh_printNormal("TestEvent: ");
5     csh_printJSON(data);
6     csh_printNormal("\n");
7     return NULL;
8 }
9
10 char *onInit() {
11     return NULL;
12 }
13
14 int main(int argc, char *argv[]) {
15     csh_InitModule(argv, argc);
16
17     csh_RegisterEvent("testevent", onTestEvent);
18
19     csh_BuildModule("TestModul", "Author", "1.0", "Das ist ein Test Modul", onInit);
20     return 0;
21 }

```

Um ein Ereignis zu auszulösen, benutzt man die Funktion `csch_callEvent`. Dabei werden 2 Parameter übergeben:

- Name | Der Name des auszulösenden Events
- Data | Die JSON Daten, die mit dem auslösen des Events übertragen werden sollen.

```

1
2 csch_callEvent("testevent", data);

```

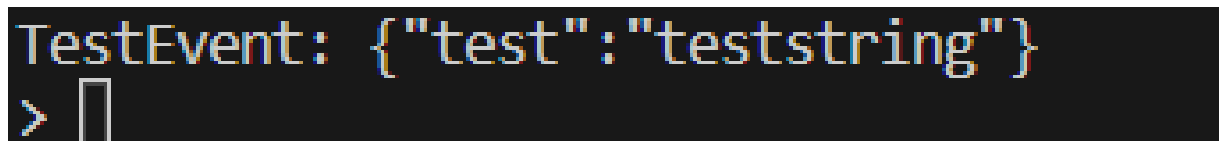


Abbildung 3: Beispiel CShell

### 3.4 Aktionen

Funktionen, die die CShell steuern können:

- `csch_printNormal` | Gibt einen normalen Text aus.
- `csch_printInfo` | Gibt einen Text als Information aus.
- `csch_printError` | Gibt einen Text als Fehler aus.
- `csch_printFatal` | Gibt einen Text als Fatalen Fehler aus.
- `csch_printJSON` | Gibt ein JSON Object als Text aus.
- `csch_Shutdown` | Schaltet die CShell aus.
- `csch_ExecuteCommand` | Führt eine Befehl auf der CShell aus -> Gibt einen Error-Code zurück.



```

1
2     void csh_printNormal(const char *format, ...);
3     void csh_printInfo(const char *format, ...);
4     void csh_printError(const char *format, ...);
5     void csh_printFatal(const char *format, ...);
6     void csh_printJSON(cJSON *json);
7
8     void csh_Shutdown(int error_code);
9     int csh_ExecuteCommand(char *str, bool silent_mode);

```

## 4 NSSH-Protokoll

Das NSSH-Protokoll(Network Secure Shell) habe ich eigens für die CShell programmiert. Dieses Protokoll benutzt **JSON**(Javascript Object Notation) und **SSL/TSL** (Secure Sockets Layer/Transport Layer Security) um mit der CShell zu kommunizieren und zu interagieren. Den Client dazu kannst du im Ordner **Utils**, im gleichen Ordner, wo sich die CShell befindet, finden. Beim Ausführen musst du einfach nur die IP und den Port(Standart: 8921) eingeben. Wenn die Fehlermeldung **Cannot connect to Server: You are banned!** kommt, heißt das, dass dein PC von der CShell gesperrt wurde. Wenn die Fehlermeldung **Cannot connect to Server: Unknown Reason!**, kann sich der Client aus nicht bekannten Gründen nicht mit dem Client verbinden.