

Second Report

Vol-TeX-Sh

May 2021



VolTeX-sh\$
not the bourn-again shell

by VolTeX Studio

BOSE Abhishek JOHN Rajat BEKHAT Sofiane

Contents

1	Overview of our Progress	1
1.1	Parsing	1
1.2	Autocompletion	1
1.3	Execution	1
1.4	Commands	1
1.5	Work in progress	2
2	Implementation	3
2.1	Parsing	3
2.2	New Commands	3
2.2.1	calc	3
2.2.2	alias	4
2.2.3	hostname	5
2.3	Ctrl+Z and Ctrl+C signals handling	6
2.4	Job	7
2.5	grep	8
3	What's Next??	10

1 Overview of our Progress

1.1 Parsing

First things first, our shell needs to be able to read whatever the user types. For this end, we have to parse the input. The method is simple : we read the input and store it in an array. After, we traverse the array and extract the strings one by one and store them in another array. Furthermore, our parsing algorithm handles the case when there are multiple spaces between strings.

Small change has been made : we are using the function `getline()` in order to read from the prompt.

Feel free to read the code in section 2

1.2 Autocompletion

Autocomplete, or word completion, is a feature in which an application predicts the rest of a word a user is typing. For this purpose, we use the function `getline` from the library `<getline/readline.h>` which has autocompletion incorporated by default.

1.3 Execution

In order to execute a command, our shell compares the command typed by the user with all possible commands. If the comparison is successful , it executes the command by calling its function. Otherwise the shell returns an error, saying that the input command does not exist.

1.4 Commands

For our shell, we have taken heavy inspiration from the command terminal used in Linux based operating systems. All the commands that we have implemented are designed to simulate an experience similar to one you have while using a terminal.

Here is list of all the commands we have implemented:

1. alias
2. calc
3. cat* (2)
4. cd
5. clear
6. color
7. echo
8. exit
9. grep
10. help
11. hostname (1)
12. job
13. ls* (1)
14. mkdir
15. mv
16. pwd
17. rm
18. rmdir* (1)
19. touch
20. tree

(* signifies that this command also includes certain attributes that can be added to modify the behaviour of the command. Number of attributes are in the parenthesis)

1.5 Work in progress

We are working on history aspect of the shell. It will be based on the signal catching and storing the history in the memory. Along with this, we are also working on the piping and fd redirection. We will be introducing the PATH variable for defence 3.

2 Implementation

2.1 Parsing

Here is the code of the parsing algorithm :

```
void read_command(char **parameters, int *nb_par)
{
    char *line;
    int sub_index = 0;
    int i = 0;
    line = readline(" ");
    int j = 0;
    while(line[j] != '\0')
    {
        if (line[j] == ' ' && sub_index != 0)
        {
            parameters[i][sub_index] = 0;
            i++;
            sub_index = 0;
        }
        else if (line[j] != ' ')
        {
            parameters[i][sub_index] = line[j];
            sub_index++;
        }
        j++;
    }
    if (line[j] == 0 && sub_index != 0)
    {
        parameters[i][sub_index] = 0;
        i++;
        sub_index = 0;
    }
    *nb_par = i;
    return;
}
```

2.2 New Commands

2.2.1 calc

This function can be used to open up a scientific calculator with up to 22 math operations. This function can be very useful to make quick mathematical calculations. It takes in input from 0 to 22. User has to input on stdin. The result is printed out on the shell. The way we have implemented the command is goes on in and infinite loop until user inputs 0.

```

Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version calc
Select your operation (0 to exit):
1. Addition      2. Subtraction  3. Multiplication  4. Division
5. Square root   6. X ^ Y       7. X ^ 2          8. X ^ 3
9. 1 / X         10. X ^ (1 / Y) 11. X ^ (1 / 3)    12. 10 ^ X
13. X!           14. %          15. log10(x)       16. Modulus
17. Sin(X)       18. Cos(X)     19. Tan(X)         20. Cosec(X)
21. Cot(X)       22. Sec(X)

Choice: 1
Enter X: 5
Enter Y: 7
Result: 12.000000

Select your operation (0 to exit):
1. Addition      2. Subtraction  3. Multiplication  4. Division
5. Square root   6. X ^ Y       7. X ^ 2          8. X ^ 3
9. 1 / X         10. X ^ (1 / Y) 11. X ^ (1 / 3)    12. 10 ^ X
13. X!           14. %          15. log10(x)       16. Modulus
17. Sin(X)       18. Cos(X)     19. Tan(X)         20. Cosec(X)
21. Cot(X)       22. Sec(X)

Choice: █

```

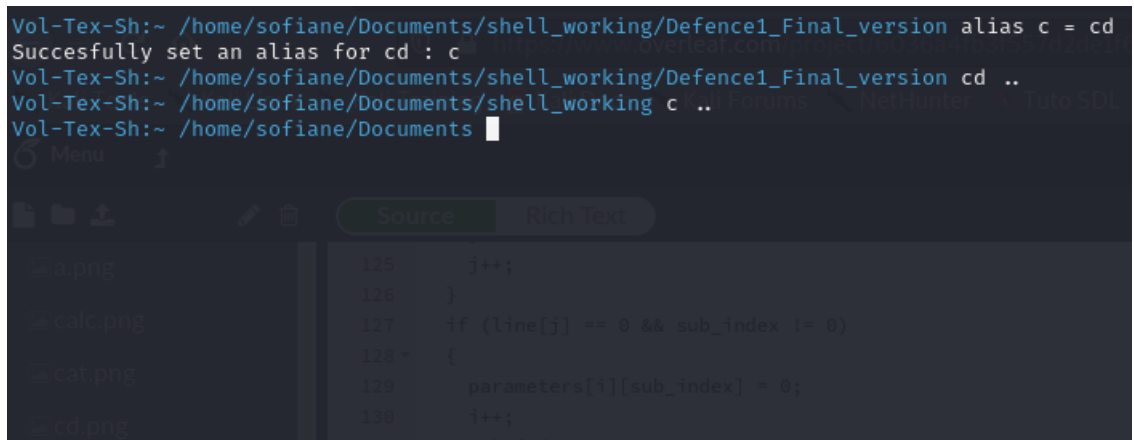
As you can see, you can perform endless number of calculations before you choose to exit the function.

2.2.2 alias

A function that helps the user interact with our shell in an easier way, this function is used to create an "alias" for any one of the existing commands. This can be used to streamline the process of using big command names by shortening them to a more comfortable command name.

The way it has been implemented is that our function tries to duplicate the 3rd argument into the 2nd argument.

```
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version alias c = cd
Succesfully set an alias for cd : c
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version cd ..
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working c ..
Vol-TeX-Sh:~ /home/sofiane/Documents
```

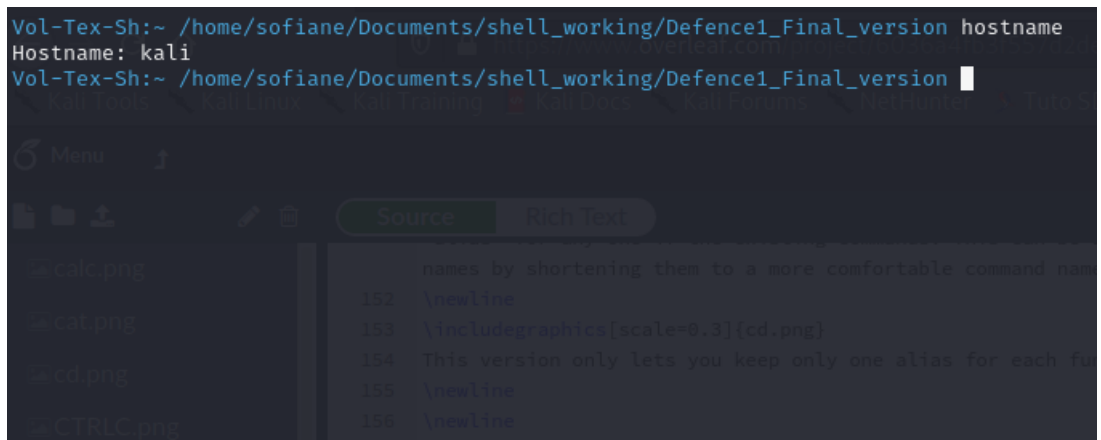


This version only lets you keep only one alias for each function, this will be improved upon in the last defense.

2.2.3 hostname

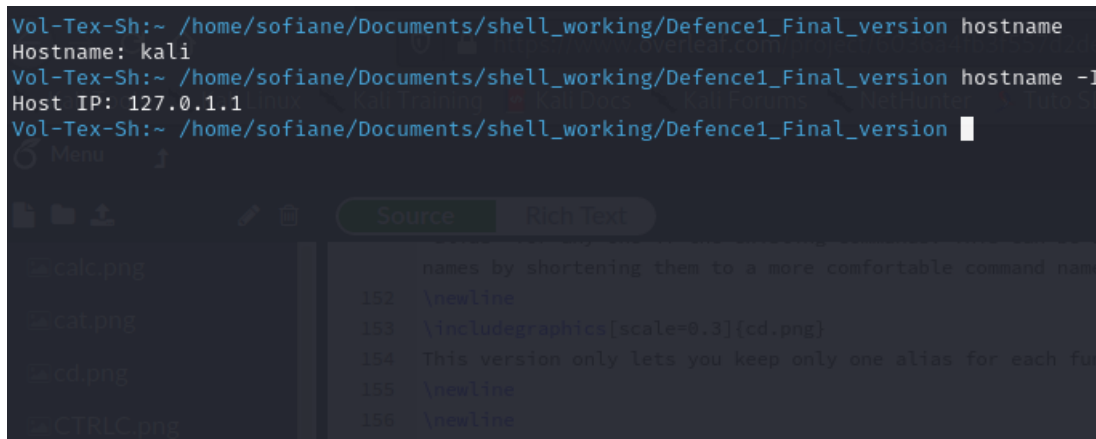
This function is used to show the system's host name. It has 2 flags to enhance the utility of this command:

```
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version hostname
Hostname: kali
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version
```



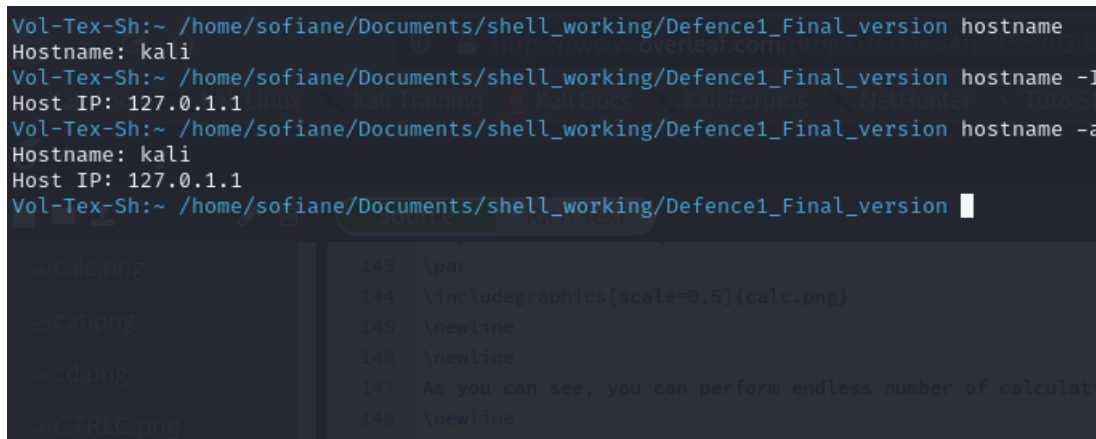
"-I" - used to display IP address of the system.

```
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version hostname
Hostname: kali
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version hostname -I
Host IP: 127.0.1.1
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version
```



"-a" - used to display all the information that this function can retrieve.

```
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version hostname
Hostname: kali
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version hostname -I
Host IP: 127.0.1.1
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version hostname -a
Hostname: kali
Host IP: 127.0.1.1
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version
```



As you can see we are working towards implementing more network based commands. This will be improved upon by the next defense.

2.3 Ctrl+Z and Ctrl+C signals handling

We have implemented the signal handling CTRL+Z which allows to put a running process in the background and suspends it. In addition , CTRL+C signal has been implemented. The latter kills a running process. We are using the library <signal.h> for this purpose.

Here is the code for CTRL+Z :

```
void ctrl_z()
{
    pid_t p = getpid();
    if (p != shellid)
```



```

        return;

    if (childpid != -1)
    {
        kill(childpid, SIGTTIN);
        kill(childpid, SIGTSTP);
        back_count++;
        back[back_count].pid = childpid;
        back[back_count].is_back = 1;
        strcpy(back[back_count].name, fore.name);
    }
    signal(SIGTSTP, ctrl_z);
}

```

Finally here is the code CTRL+C:

```

void ctrl_c()
{
    pid_t p = getpid();
    if (p != shellid)
        return;
    if (childpid != -1)
        kill(childpid, SIGINT);
    signal(SIGINT, ctrl_c);
}

```

As one can see, we are using the signals in order to catch the CTRL+C and CTRL+Z signals.

2.4 Job

Jobs command is used to list the jobs that you are running in the background and in the foreground. If the prompt is returned with no information no jobs are present. In order to implement this , we defined a job struct :

```

typedef struct job
{
    char name[100];
    int pid;
    int is_back;
} job;

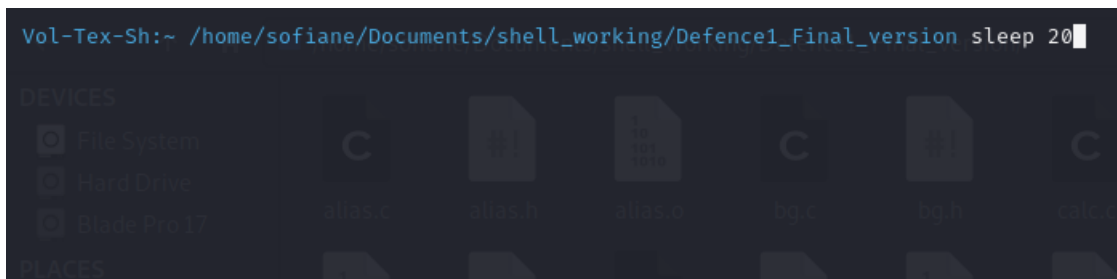
```

Thanks to this struct, we are able to keep track of all the running processes and store their process IDs.

Now, let's run this command and see how it works:

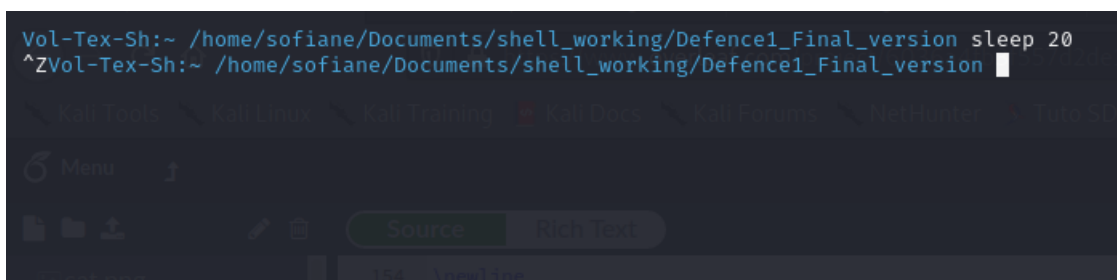
-1- Let's run the sleep command for 20 seconds.

```
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version sleep 20
```



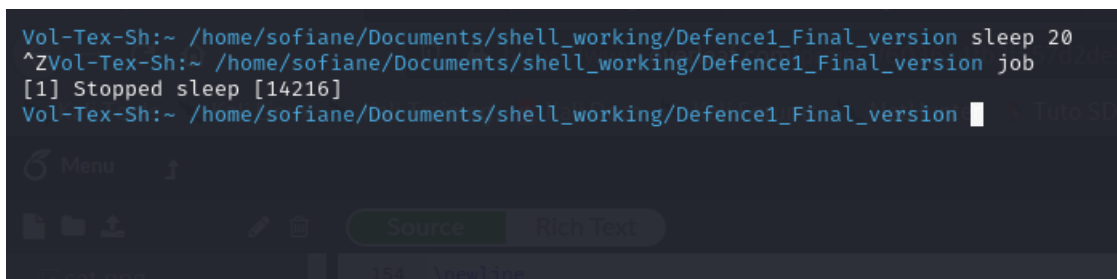
-2- Now let's suspend the process by triggering the CTRL+Z signal.

```
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version sleep 20
^ZVol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version
```



-3- Now let's run the job command.

```
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version sleep 20
^ZVol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version job
[1] Stopped sleep [14216]
Vol-TeX-Sh:~ /home/sofiane/Documents/shell_working/Defence1_Final_version
```



As one can see, the sleep process has been stopped and it became a background process. Therefore one can use the shell again.

2.5 grep

Grep basically is a command that tries to match the pattern which could be a regular expression or a sub-string to a file. It could be multiple files. I use regex to do grep.

A regular expression is a sequence of characters that specifies a search pattern. Usually such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation. It is a technique developed in theoretical computer science and formal language theory.

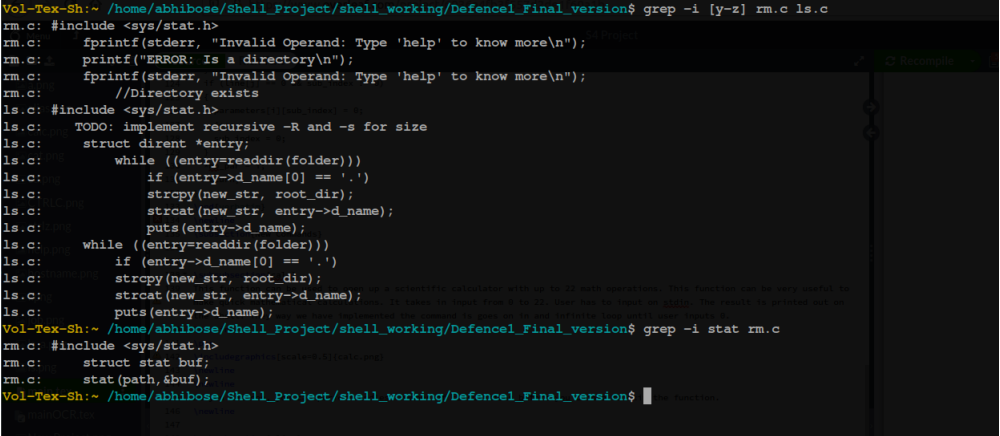
For this we have used the regex library. The implementation and usage of the regex library was kinda difficult to understand with all the flags. But basically we have implemented two attributes for grep, -i and -E. -i is a ignore case flag which does not recognise Extended regular expression. -E is the attribute to use Extended regular expressions.

```
void compile_pattern(const char *pat)
{
    int flags = REG_NOSUB; /* don't need where-matched info */
    int ret;
    #define MSGBUFSIZE 512 /* arbitrary */
    char error[MSGBUFSIZE];

    if (ignore_case)
        flags |= REG_ICASE;
    if (extended)
        flags |= REG_EXTENDED;

    ret = regcomp(& pattern, pat, flags);
    if (ret != 0) {
        (void) regerror(ret, & pattern, error, sizeof error);
        fprintf(stderr, "%s: pattern '%s': %s\n", myname, pat, error);
        errors++;
    }
}
```

As you can see, this code is for compiling the pattern, either a regular expression or a substring. It uses regcomp function from the library which basically a buffer as the first argument and stores the matched info. After it has the compiled pattern, it goes through the lines of the file and matches the string.



```
Vol-TeX-Sh:~ /home/abhibose/Shell_Project/shell_working/Defence1_Final_version$ grep -i [y-z] rm.c ls.c
rm.c: #include <sys/stat.h>
rm.c:     fprintf(stderr, "Invalid Operand: Type 'help' to know more\n");
rm.c:     printf("ERROR: Is a directory\n");
rm.c:     fprintf(stderr, "Invalid Operand: Type 'help' to know more\n");
rm.c:     //Directory exists
ls.c: #include <sys/stat.h>
ls.c:     TODO: implement recursive -R and -s for size
ls.c:     struct dirent *entry;
ls.c:     while ((entry=readdir(folder)))
ls.c:         if (entry->d_name[0] == '.')
ls.c:             strcpy(new_str, root_dir);
ls.c:             strcat(new_str, entry->d_name);
ls.c:             puts(entry->d_name);
ls.c:     while ((entry=readdir(folder)))
ls.c:         if (entry->d_name[0] == '.')
ls.c:             strcpy(new_str, root_dir);
ls.c:             strcat(new_str, entry->d_name);
ls.c:             puts(entry->d_name);
Vol-TeX-Sh:~ /home/abhibose/Shell_Project/shell_working/Defence1_Final_version$ grep -i stat rm.c
rm.c: #include <sys/stat.h>
rm.c:     struct stat buf;
rm.c:     stat(path,&buf);
Vol-TeX-Sh:~ /home/abhibose/Shell_Project/shell_working/Defence1_Final_version$
```

3 What's Next??

For the next defence we are gonna implement piping and fd redirection. We realise by this defence, it should have been done. But because of internal problems in the group, we lagged behind. Nevertheless, we will strive to finish our job. Along with this we are gonna implement the history of the shell. Few more commands. And the regex will be implemented for every command line. We will be introducing the GUI and the PATH variable. We feel we have the basics ready for a shell. We just need to wrap up few things.

At the end, we would like to say that we are glad with whatever progress we have made so far considering the problems we were facing with one of our teammates who is no more in our group.