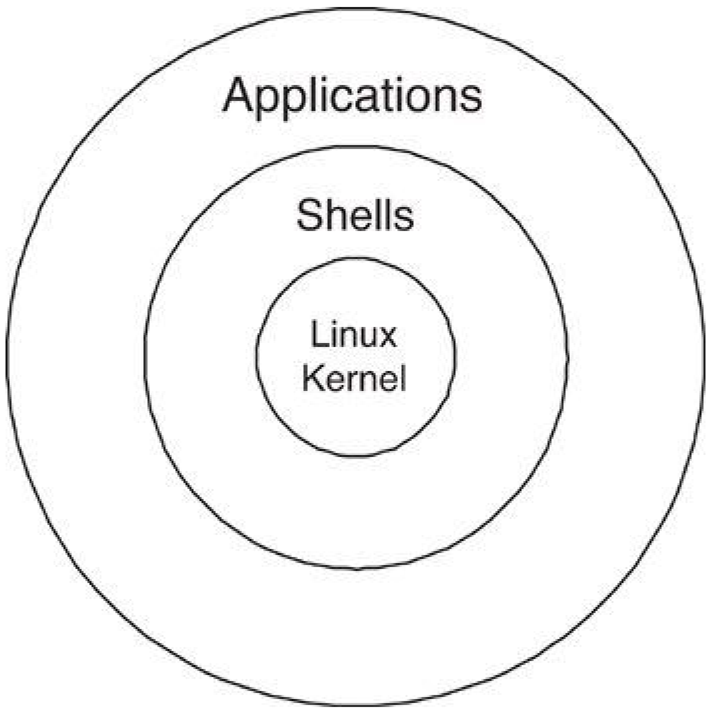
USP-Project 2 Shell



TEAM

1.	Sanketh Rangreji	-	01FB15ECS267
2.	Saurav Sachidanand	-	01FB15ECS272
3.	Shahid Ikram	-	01FB15ECS274

OVERVIEW:

A **Shell** provides with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

IMPLEMENTATION

We have implemented a basic shell which intakes the commands from the user.

We preprocess the command that has been entered by the user.

For Preprocessing we have used String Processing in C.

For every command entered by the user a child is forked. The child memory image is replaced with that of the command that has to be executed with the help of execvp command.

We have chosen the execvp of the family of exec calls.

- ❖ The execvp takes the arguments as a vector which is terminated by NULL.
- ❖ It searches for the executable in the path specified by the $PATH env. variable

FEATURES

We have supported all the **basic functions** like:

1. ls
2. ps
3. cat
4. echo
5. cd .....

Along with these we have implemented the **history** command, which gives the history of all commands that have been executed along with their timestamp.

We have also implemented the **alias** functionality where we can define the alias for a command and then use the alias in place of the command.

```
$ alias ll = ls -l
```

The command **'aliases'** will list the the aliases set so far.

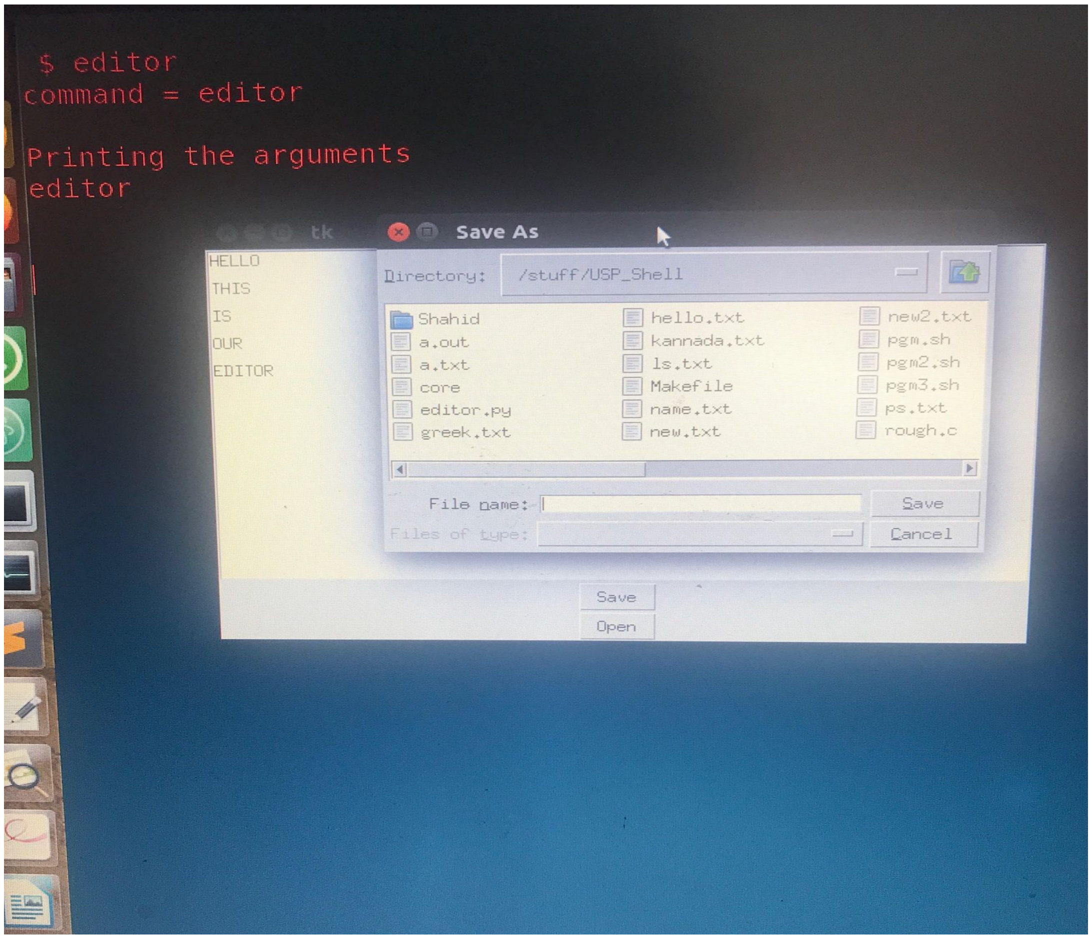
```
$ aliases
```

We have supported **input and output redirection**.

We have also supported **piping to one level depth**.

We have implemented an **editor** as well. The editor opens with the command

```
$ editor
```



```
 $ editor
command = editor

Printing the arguments
editor
```

In the editor we can create new files, save them and also open and modify existing files.

We have implemented the editor in Python using the tkinter module.

We can execute **shell scripts** by just specifying the path of the script.

$ ./script.sh

We have also added this feature where the user can **execute one of the previously executed commands** from the history buffer.

The user can specify a '%' followed by a number to execute one of the previous commands indicated by the number. For example:

$ %3 - This command executes the 3rd most recent command.

CUSTOM FUNCTIONS

We have implemented 3 custom functionalities for the shell that aren’t directly supported **open-file <filename with extension>**

a. OPENING A FILE WITH ITS DEFAULT APPLICATION
b. Usually to open any file we need to specify the application with which we want to launch it.
c. For Example: To open a ‘.py’ file we need to specify the editor as well.
    i. $ gedit a.py
    ii. $ subl a.py
    iii. $ atom a.py
d. But when we double click on the file through the GUI we needn’t specify the application with which we want to open it every time.
e. We included this feature into our shell where when a user wants to open any file he need not specify which application to use. Instead the file gets opened with the default application with which it is supposed to be open.
f. For Instance:
    i. If we want to open a video file

    ```
    $ open-file <video file name with extension>
    ```

    ii. If we want to open an audio file

    ```
    $ open-file <audio file name with extension>
    ```

2. **open-prev-file <N>**
    a. All the files that are opened using open file are maintained and we can then use this command to open Nth previous file opened by open-file.
    b. For instance:
        i. This opens the 3rd most recently opened file

```
$ open-prev-file 3
```

3. **translate <text file to translate> <target-language>**

TRANSLATING A GIVEN FILE(TEXT) TO A SPECIFIED LANGUAGE

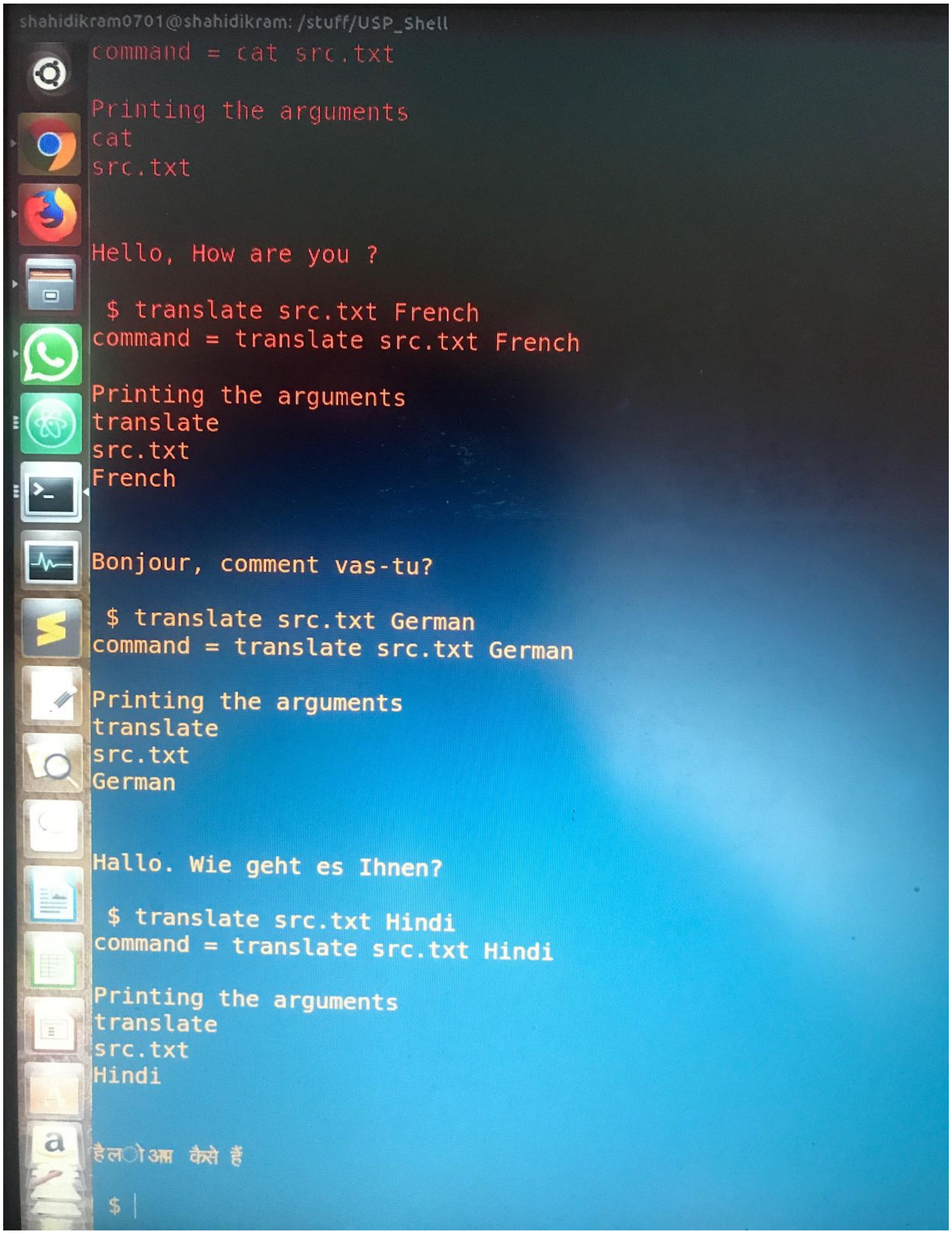
   a. The user can translate a given text file in English to any other specified language, the command is given as follows:

```
$ translate <name_of_text_file> <target_language>
```

   It should be noted that the translation is only one way **from English to any other language supported by the API.**

   The way this works is that a python script that uses the Google Translate API is called when the above command is parsed by our shell . Hence the support for languages and accuracy of the translation is dependent on the API, and will throw an error message if an invalid target language is specified. For languages that use a different character set than English(like Hindi or Greek or Russian) , the translated output is dependent on the system's support for the specific character set of a given language.

The translate command in action :



```
shahidikram0701@shahidikram: /stuff/USP_Shell
command = cat src.txt

Printing the arguments
cat
src.txt


Hello, How are you ?

 $ translate src.txt French
command = translate src.txt French

Printing the arguments
translate
src.txt
French


Bonjour, comment vas-tu?

 $ translate src.txt German
command = translate src.txt German

Printing the arguments
translate
src.txt
German


Hallo. Wie geht es Ihnen?

 $ translate src.txt Hindi
command = translate src.txt Hindi

Printing the arguments
translate
src.txt
Hindi


हेलोआप कैसे हैं

 $ |
```