# Basic UNIX Shell with Python3

## Author: Sergey Platonov

ITI 3rd semester, HEPIA, Geneva System Dev Class with Adrien Lescourt

# Introduction

This project was done as a part of the SysDev class taught by Adrien Lescourt. The goal was to gain a deeper understanding of the UNIX Shell by implementing a basic version of it ourselves.

# Setup

The Shell is coded with Python3. Import the project with the following command:

```
git clone https://github.com/maganoegi/PyTerm.git
```

Next, you shall see the following files:

```
* commands.py           # bash commands definitions
* inputHandler.py       # input parsing functions definitions
* pyUnix.py             # main driver file
* README.md             # this file
* std.py                # config-style file with stdin, stdout, stderr
globals
```

To start the shell, use execute **pyUnix.py** with python3:

```
python3 pyUnix.py
```

You will see the following prompt (here used with **ls -l** command):

```
Unix-like shell project
Author:          Sergey Platonov
Professor:       Adrien Lescourt
2019-2020       HEPIA, Geneva, CH       ITI 3 semester
For supported commands, type all-cmd
__________________________________________

?> ls -l
-rw-r--r-- README.md
```

```
-rw-r--r-- std.py
-rw-r--r-- .platonovrc
-rw-r--r-- pyUnix.py
-rw-r--r-- inputHandler.py
-rw-r--r-- commands.py

?>
```

# General Code Organisation

The information flows in the following fashion:

```

    +------------------------+
    | pyUnix.py              |
    +------------------------+
    | while True:            |
    |    wait for user input |
    |    split into words    |
+------+ parse words         |
|   |    print stdout/stderr <-----------------------+
|   +------------------------+                       |
|                                                    |
|                                                    |
|   +-------------------------------------------+    |
|   | inputHandler.py                           |    |
|   +-------------------------------------------+    |
|   | for each word:                            |    |
|   |   check if aliases exist                  |    |
|   |   check whether a word is a command       |    |
+--->      execute command                      |    |
|   |      update global stdin, stdout, stderr  <----+
|   +-------------------------------------------+    |
|                                                    |
|                                                    |
|                                                    |
|   +--------------------+            +---------+    |
|   | commands.py        |            | std.py  |    |
|   +--------------------+            +---------+    |
+--->      command       +------------>  stdin  +----+
    +--------------------+      +----->  stdout |---+
                                 +---->  stderr +--+
                                      +---------+
```

Another file, **.platonovrc**, imitates the function of **.bashrc**. It is generated upon first execution of the script, and contains *aliases* in a **json** format.

The general idea is: a line is decomposed into "words". These words then are parsed one by one, checking whether they are a valid command. If "flags" are possible, the "words" that follow are checked as well, in order to include them into the expression. If at any point an error is encountered, **stderr** is written to and the program stops - not without displaying what the error is! If a command is run successfully, the output is written to the **stdout**.

If a **;** separator is present, line is decomposed into lines. Then we process each line as mentioned above.

Once a line has been processed without fail and the resulting **stdout** is printed onto the terminal (provided it's not empty).

The library used for system operations is python's **os** library.

# Reflections

## Difficulties

I did not have enough time to implement the **pipe** functionality. Due to time constraints I decided to do the rest of the functionalities well enough, and leave piping to the last - seen that it's relatively simple and requires funnelling of stdout into stdin, which then needs to be taken as an input if it's not empty.

Another nice thing I would have loved to do is to think about a way to avoid the huge if-elif-else sequence in inputHandler.py. Since switch-case does not exist in Python, this was the best option at the time. Maybe POO functionality could have provided a better solution. To be explored.

Noteworthy commands were: **duplicate** for the search for the optimal way to find matches, **wc** for flag handling, **rm/mv/touch** for my first experience with callbacks in python, and ofcourse **tree**, which was my favorite part of this project. I wanted to have it exactly like in bash, and I managed to achieve that.

I am a bit dissapointed that I did not have enough time to finish the project entirely (pipe, smarter separation with 😉, but I have learned A LOT for the projects to come. And since Python has become my go-to language, this experience is very valuable.

## Conclusion

Very enriching project, best way to understand what is happening behind the schemes of a shell, even though the project's scope is limited. It's a pleasure to work with Python, and I would love to master this language entirely.