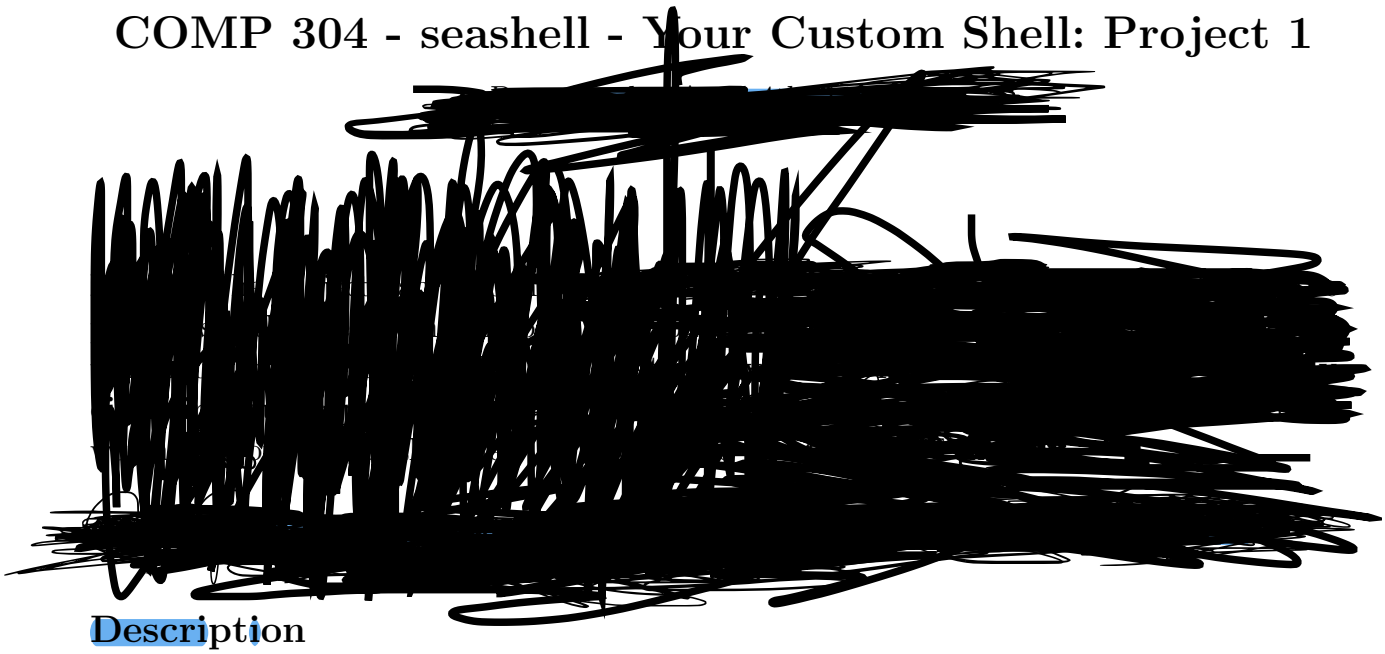
COMP 304 - seashell - Your Custom Shell: Project 1



Description

This project aims to familiarize you with the Unix system call interface and the shell by letting you implement several features in a shell, called **seashell**, using C/C++. A shell is a program that provides interface to the operating system by gathering input from the user and executing programs based on that input. Once executed, the **seashell** will read both system and user-defined commands from the user. The projects has six parts:

Part I

(20 points)

- Use the skeleton program provided as a starting point for your implementation. The skeleton program reads the next command line, parses and separates it into distinct arguments using blanks as delimiters. You will implement the action that needs to be taken based on the command and its arguments entered in **seashell**. Feel free to modify the command line prompt and parser as you wish.
- Read *"Project 1- Unix Shell"* at the end of Chapter 3 in the book starting from P12 (10th edition). That will be useful.
- Use **execv()** system call (instead of **execvp()**) to execute common Linux programs (e.g. ls, mkdir, cp, mv, date, gcc) and user programs by the child process. The difference is that **execvp()** will automatically resolve the path when finding binaries, whereas for **execv()** your program should search the path for the command invoked, and execute accordingly.

Part II

(10 points) **shortdir**: In this part, you will implement a new command, **shortdir**, to associate short names with the current directory. The purpose is to reach the directory (change

to the directory) with a short name instead of typing the whole path. For instance, we associate the name *matlab* with the directory */usr/local/MATLAB/R2018b/* and use it to change to that directory by using the name *matlab*. You will also implement supportive options for the *shortdir* as follows:

- *shortdir set name*: associates *name* with the current directory. Overwrites an existing association.
- *shortdir jump name*: changes to the directory associated with *name*.
- *shortdir del name*: deletes the name-directory association.
- *shortdir clear*: deletes all the name-directory associations.
- *shortdir list*: lists all the name-directory associations.

Note that this command **lives across shell sessions**; when a new shell session is started, it should remember the associations from the previous sessions. Refer to Listing 1 for sample usage of shortdir.

Listing 1: Setting alias for a directory using shortdir

```
seashell> pwd
/usr/local/MATLAB/R2018b
seashell> shortdir set matlab
matlab is set as an alias for /usr/local/MATLAB/R2018b/
...
seashell> shortdir jump matlab
seashell> pwd
/usr/local/MATLAB/R2018b
```

Part III

(10 points) **highlight**: In this part, you will implement the *highlight* command that takes a word-color pair and a text file as an input. For each instance of the word appearing in the text file, the command prints the line where the word appears and highlights the word with that color. The colors can be one of the r (red), g (green) or b (blue) colors. The word check is not case-sensitive. The command syntax and its sample usage are as shown below:

```
seashell> highlight <word> <r | g | b > <filename>
seashell> highlight language r file.txt
The programming language used for this code is C.
The first three letters of English language are A B and C.
```

Part IV

(15 points) In this part of the project, you will implement a new **seashell** command, *goodMorning*:

- This command will take a time and a music file as arguments and set an alarm to wake you up by playing the music using rhythmbox. In order to implement *goodMorning*, you may want to use the crontab command.

Listing 2: Sample usage of goodMorning command

```
seashell> goodMorning 7.15 /home/musics/muse.mp3
```

Before implementing the new command inside **seashell**, you should get familiar with crontab (if you decide to use crontab) and rhythmbox on a regular shell.

More info about rhythmbox:
http://manpages.ubuntu.com/manpages/hardy/man1/rhythmbox-client.1.html
More info about crontab:
http://www.computerhope.com/unix/ucrontab.htm

Part V

(20 points) In this part, you will implement the *kdiff* utility to compare two files in given paths. Sample usage of *kdiff* to compare two files is given in Listing 3. The utility will operate in two modes:

- $-a$: the utility reads the input files (*file1.txt* and *file2.txt*) as text files and compares them line-by-line. If the two files are different, the utility first prints the differing lines from each file and then prints the count of differing lines as shown in Listing 3. The utility checks for file extensions and if they are not .txt, flags an error. If the two files are identical, it displays the message "The two files are identical".

- $-b$: the utility reads the input files as binary files and compares them byte-by-byte. If the two files are different, the utility prints the message "The two files are different in xyz bytes", where xyz is the number of bytes different between the files. The utility accepts files with any extension in this case. If the files are identical, the message saying "The two files are identical" is displayed. Refer to Listing 3 for the sample output.

  If the user does not provide either $-a$ or $-b$, $-a$ is assumed by default.

  Note: You are not allowed to use existing file comparison tools (such as *diff*) to compare the files. We expect you to use C/C++ file I/O operations to complete this task. You may use the existing tools to verify your output.

Listing 3: *kdiff* utility sample output

```
seashell> kidff -a file1.txt file2.txt
The two text files are identical

```

```
seashell> kdiff -a file1.txt file2.txt
file1.txt:Line 23: A quick brown fox jumps over the lazy dog
file2.txt:Line 23: A swift squirrel climbs the tree
file1.txt:Line 41: Corona cases today = 1200
file2.txt:Line 41: Corona cases today = 689
2 different lines found

seashell> kdiff -a file.txt file2.txt
The two files are identical

seashell> kdiff -b file1.bin file2.bin
The two files are identical

seashell> kdiff -b file1.bin file21.bin
The two files are different in 8213 bytes
```

Part VI

(10 points) In this final part, you will implement a command of your choice in **seashell**. Come with a new, non-trivial command and implement it inside **seashell**. Creativity will be rewarded and selected commands will be shared with your peers in the class. It is possible that command you come up with is already implemented in Unix. That should not stop you from implementing your own but do not simply execute the existing Unix version of it.

Deliverables

You are required to submit the followings packed in a zip file (named your-username(s).zip) to the blackboard :

- .c source code file that implements the **seashell** shell. Please comment your implementation.
- any supplementary files for your implementations (e.g., Makefile)
- a short REPORT file briefly describing your implementation, particularly the new command you invented in Part V. You may include your snapshots in your report.
- Do not submit any executable files (a.out) or object files (.o) to blackboard. Do not submit the file for the song you use in Part-III.
- You are required to implement the project on a Unix-based virtual machine or Linux distribution.
- (15 points) Finally there will be project demos after the deadline. Demos account for 15 points towards your grade. We expect that both team members have equally contributed to the project and are competent to answer questions on any part of the implementation. TA will direct questions to any members of the team thus team members may receive different grades based on their demo performance.

GOOD LUCK.