Project 3

by Theint New Nyein, Maria Colon, Leo Goldstein, Josh Moore

This project was a little more difficult for the group as adding many more commands than previously, having to handle special characters, a keyword and Pipelining was intensive. Due to the large amount of work, we decided to split the work based on the complexity the work would be, as well as the capability of the individual.

Theint worked on all the internal commands, implementing the new commands the program would handle, Leo worked on the Special characters, having to handle the finding of the input based on the character received, Maria worked on the keyword of IF-Then-Else, and finally Josh worked on pipelining the previous output into a secondary input.

Internal Commands:

Alias and unalias commands (command_alias.cpp) Command_alias.cpp implements two internal commands:

alias and unalias, which have similar functionalities as the Unix-like commands to add and remove custom shortcuts for long commands. It consists of two functions: ‘internal_alias’ and ‘internal_unalis’ to handle adding and removing shortcuts, respectively. Both functions utilize string manipulation techniques to extract relevant information from the user input and interact with the aliasMap data structure Alias has the following functionalities in our program:

    -h: Display help message for the alias command.
    -p: Display all existing aliases.
    -a: Remove all aliases.

alias_name='command': Define an alias where alias_name is the name of the alias, and command is the associated command

Unalias has the following functionalities:

    -h: Display a simple help message for the unalias command.
    alias_name: The name of the alias to remove.

Help command (command_help.cpp)

The Help command in command_help.cpp provides users with assistance and information about various internal commands supported by the program. It displays a general help message when invoked without any arguments. This message provides an overview of the available internal commands and how to

access their respective help messages. In addition, users can request help for specific commands by providing the command name as an argument to the Help command. For each supported command (e.g., cd, alias, unalias, history, set), the Help command provides detailed information on its usage, options, and functionalities.

History command (command_history.cpp) The History command in command_history.cpp allows users to manage and view the command history of the program. It provides functionalities to display, clear, delete specific entries, and retrieve past commands from the command history. Additionally, users can reference the most recent command or a command by its index using special notations. It has the following functionalities:

-h: Display a simple help message for the history command.

-c: Clear the command history by deleting all entries.

-n NUM: Display the last NUM commands in history.

-d offset Delete the history entry at position OFFSET. Negative offsets count back from the end of the history list.

!n Retrieve the most recent command from the history list.

!! Retrieve the nth command from the history list.

Set command (command_set.cpp)

The Set command in command_set.cpp offers users a versatile tool for configuring shell options and environment variables within the program. It leverages the functionality of the execlp system call to execute shell commands, enabling users to modify various settings and behaviors dynamically. It serves just like Unix's set command, and the following are a few of its functionalities:

Help Message Display (-h): Users can access a detailed help message for the Set command by specifying the-h option.

Configuration of Shell Options: The Set command enables users to configure shell options by providing specific command-line arguments. For example, users can set options such as-o,-e,-u, among others, to modify shell behavior and control error handling, environment variable expansion, and other shell functionalities.

Keywords:

Maria oversaw implementing the keywords IF-THEN, ELSEIF, ELSE, and ENDIF. This was a very difficult task, to start there was a misunderstanding on how exactly the keywords were supposed to work. At first, we thought they worked like the bash version of the IF-THEN ELSEIF, so initial coding with this in mind, however, that line of thinking was incorrect. In actuality, the way the project had described the keywords was not only easier but made a lot more sense. In short, if the IF statement is true run the next command, if it

is false and there is an ELSEIF treat it like an IF, statement, if there is not an ELSEIF, but there is an ELSE should the IF is false then jump to the ELSE. the ENDIF ends the whole statement and must be included. This caused me to restart the external commands. The new approach involved splitting the input given, then finding the commands within the string. Substrings were used to facilitate the process. Having to account for running external commands within the keywords some of the logic had to be readjusted for it to work properly. There were also some bugs that needed to be fixed, to start the ENDIF was not being recognized because we forgot to account for the space. Once that was fixed the program ran, another bug that was caught, only getting single characters form the sub string, this issue was fixed by changing the end point of the subtring. We won't list allow for the bugs that were found just the ones, that were the most difficult.

Special Characters:

The hardest part was figuring out how to detect the closed brackets "]". I had to do quite a bit of research in order to succeed. Eventually Leo got it working after finding out about strchr. From there it was straightforward, just have it return something depending on what special character it detects. Beyond that was relatively easy for detection.

Pipeline:

Pipelining was a little difficult to get setup, but having only certain commands and actions account for the pipeline meant that some external usage could be implemented to handle it. For example, if you wanted to input a string into a file, you can use the 'echo' command, then pipeline it into a file. Some external handling helped get this to work.

Code:

```
Tesla: [~/SP24/Operating_Systems/Project_3] $ make
mkdir object_files
gcc -c implement_files/command_cd.cpp -o object_files/command_cd.o
gcc -c implement_files/command_help.cpp -o object_files/command_help.o
gcc -c implement_files/runnable.cpp -o object_files/runnable.o
gcc -c implement_files/special_characters.cpp -o
object_files/special_characters.o
gcc -c implement_files/placholder_file.cpp -o object_files/placholder_file.o
gcc -c implement_files/command_history.cpp -o object_files/command_history.o
gcc -c implement_files/command_set.cpp -o object_files/command_set.o
gcc -c implement_files/history_manager.cpp -o object_files/history_manager.o
gcc -c implement_files/keywords.cpp -o object_files/keywords.o
gcc -c implement_files/command_alias.cpp -o object_files/command_alias.o
gcc -c implement_files/util.cpp -o object_files/util.o
g++ main.cpp -o gamma ./object_files/command_cd.o
./object_files/command_help.o ./object_files/runnable.o
./object_files/special_characters.o ./object_files/placholder_file.o
./object_files/command_history.o ./object_files/command_set.o
./object_files/history_manager.o ./object_files/keywords.o
./object_files/command_alias.o ./object_files/util.o
Tesla: [~/SP24/Operating_Systems/Project_3] $ make run
./gamma
Entering interactive mode...
/home/josmoor/SP24/Operating_Systems/Project_3 $ if alias newLS='ls -ld';
then echo "External command with Keyword"; endif
Alias 'newLS' set to ''ls -ld''

"External command with Keyword"
/home/josmoor/SP24/Operating_Systems/Project_3 $ ls -ld
drwxr-xr-x 9 josmoor domain users 4096 Mar 28 12:45 .
/home/josmoor/SP24/Operating_Systems/Project_3 $ unalias newLS
Alias 'newLS' removed.

/home/josmoor/SP24/Operating_Systems/Project_3 $ set
VSCODE_GIT_ASKPASS_MAIN=/home/josmoor/.vscode-
server/bin/863d2581ecda6849923a2118d93a088b0745d9d6/extensions/git/dist/askpa
ss-main.js
XDG_SESSION_CLASS=user
XDG_SESSION_TYPE=tty
SHELL=/bin/bash
_=/usr/bin/make
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
KRB5CCNAME=FILE:/tmp/krb5cc_608083365_665w8m
SSH_CONNECTION=68.58.62.177 59962 134.68.51.14 1294
PATH=/home/josmoor/.vscode-
server/bin/863d2581ecda6849923a2118d93a088b0745d9d6/bin/remote-
```

```
cli:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/
usr/local/games:/snap/bin:/home/josmoor/.local/bin:/home/josmoor/bin
XDG_RUNTIME_DIR=/run/user/608083365
HOME=/home/josmoor
PWD=/home/josmoor/SP24/Operating_Systems/Project_3
VSCODE_IPC_HOOK_CLI=/run/user/608083365/vscode-ipc-dc3403ed-5657-4d64-a0ed-
78869a5575c5.sock
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/608083365/bus
LOGNAME=josmoor
VSCODE_GIT_IPC_HANDLE=/run/user/608083365/vscode-git-088832d3b9.sock
COLORTERM=truecolor
SHLVL=1
XDG_SESSION_ID=1369
USER=josmoor
OLDPWD=/home/josmoor
VSCODE_GIT_ASKPASS_EXTRA_ARGS=
TERM_PROGRAM=vscode
VSCODE_GIT_ASKPASS_NODE=/home/josmoor/.vscode-
server/bin/863d2581ecda6849923a2118d93a088b0745d9d6/node
MAKEFLAGS=
MFLAGS=
SSH_CLIENT=68.58.62.177 59962 1294
TERM_PROGRAM_VERSION=1.87.2
MAKE_TERMOUT=/dev/pts/16
BROWSER=/home/josmoor/.vscode-
server/bin/863d2581ecda6849923a2118d93a088b0745d9d6/bin/helpers/browser.sh
GIT_ASKPASS=/home/josmoor/.vscode-
server/bin/863d2581ecda6849923a2118d93a088b0745d9d6/extensions/git/dist/askpa
ss.sh
MAKE_TERMERR=/dev/pts/16
LANG=en_US.UTF-8
TERM=xterm-256color
MOTD_SHOWN=pam
MAKELEVEL=1
/home/josmoor/SP24/Operating_Systems/Project_3 $ exit
```