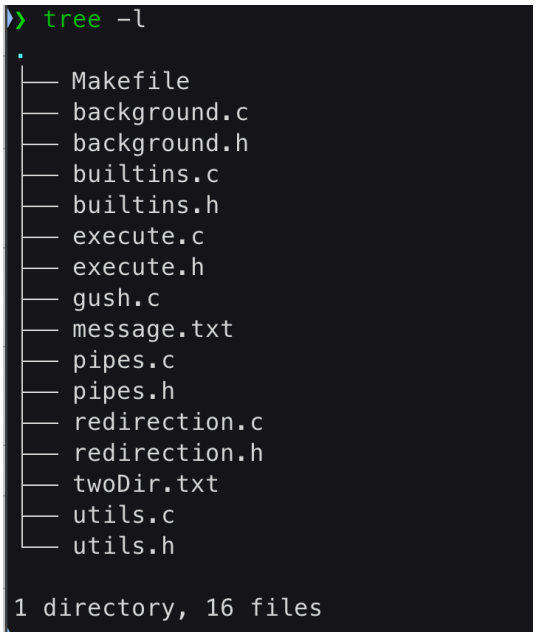
COSC3020-PA2-Pseudo Shell

Created by Cameron Daly & Jed Henry

Files Submitted.

All-test commands from “guShell_Example_Tests” PASSED



We created a Pseudo shell named gush which is our implementation of our shell in the Linux terminal. To compile the shell a Makefile was created to compile all of our files together to finally execute our gush. To begin to use the gush after the compilation of the makefile we use the ./gush command. When we want to use batch mode, batch mode uses a command that is used to read in commands from a standard input or a specified file. We use the file twoDir.txt to run the argument, so we would type in our terminal ./gush twoDir.txt. Batch mode allows gush to execute a series of commands from a specified file instead of reading input interactively. When running ./gush twoDir.txt, the shell reads commands from twoDir.txt line by line and executes them sequentially as if they were entered manually. This enables automated execution of multiple commands without user intervention. If an invalid command is encountered, gush will handle errors accordingly and continue processing the remaining commands. Batch mode is useful for scripting and testing command sequences efficiently.

Batch file processing

```
gush> ./gush twoDir.txt
Executing command: /bin/mkdir
Executing command: /bin/mkdir
Executing command: /bin/mkdir
Executing command: /bin/touch
Executing command: /bin/touch
Executing command: /bin/touch
Executing command: ./gush
gush> _
```

Basic user interaction

Simple command - /bin/ls

```
gush> /bin/ls
Makefile      background.o  builtins.o  execute.o  gush.o       pipes.c  redirection.c  test.txt    utils.c
background.c  builtins.c    execute.c   gush       message.txt  pipes.h  redirection.h  tmp.txt     utils.h
background.h  builtins.h    execute.h   gush.c     output.txt   pipes.o  redirection.o  twoDir.txt  utils.o
Executing command: /bin/ls
gush>
```

Simple command with arg - /bin/ps -aux

```
cosc3020    2669  0.0  0.0   2272  1280 tty1     S+   04:52   0:00 ./gush
root        2675  0.0  0.0      0     0 ?        I    04:52   0:00 [kworker/2:0]
cosc3020    2685  0.0  0.0   8820  3968 tty1     R+   04:56   0:00 /bin/ps -aux
Executing command: /bin/ps
gush>
```

Single command with multiple arg - /usr/bin/nl -v 500 -i 2 message.txt

```
gush> /usr/bin/nl -v 500 -i 2 message.txt
   500  LINE    1       Peace on earth
   502  LINE    2       Joy to the world
   504  LINE    3       hack free or die
Executing command: /usr/bin/nl
gush>
```

Gushell consists of these files to run:

background.c/h

This process of gush handles background execution of commands using the & operator. When a user runs a command with &, it is executed in parallel, and the shell does not wait for it to complete before continuing. Instead, gush tracks

background processes and periodically checks for their completion. Inside our file we have:

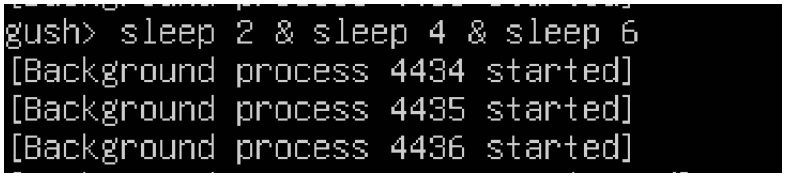
The adding Background Processes (add_background_process) stores the pid of a newly created background process in a linked list. This
ensures that all background processes can be tracked while they run.

Checking for Completed Processes (check_background_processes) uses waitpid(-1, &status, WNOHANG) to check if any background process has finished. If a process has terminated, it prints a message and removes the process from tracking. Prevents zombie processes from lingering in the system.

Cleaning Up Completed Processes (cleanup_background_process) searches the linked list for the finished process. Removes the process from the list and frees allocated memory. Ensures efficient memory management.

Process Control
Background process - sleep command

Multiple background processes - 3 sleep commands



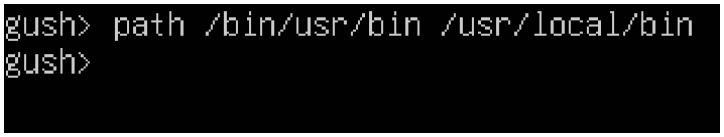
builtins.c/h

gush includes several built-in commands that are executed directly within the shell. When a user enters a command, gush first checks if it is a built-in command, the shell executes the command internally. This improves efficiency by avoiding unnecessary process creation and allows the shell to manage tasks like directory changes and tracking the history of recent commands. Commands include:

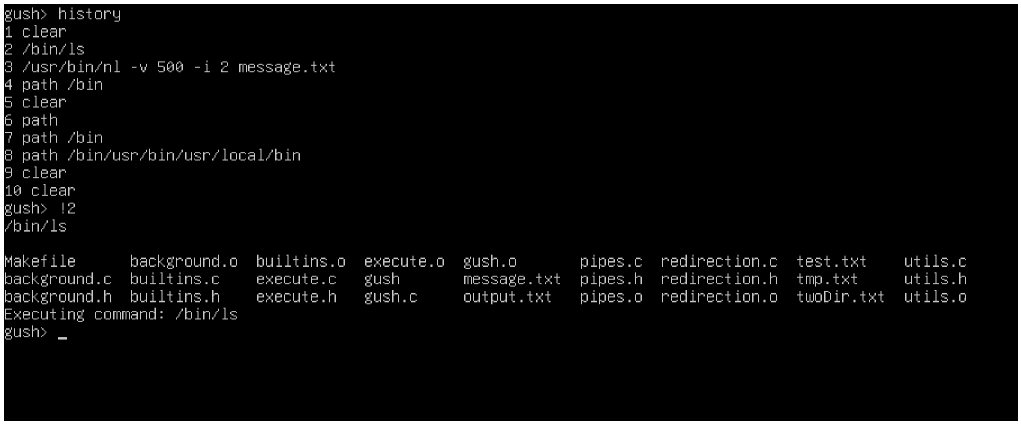
- exit(0) to terminate the shell and if additional arguments are provided, an error message is displayed.
- clear to refresh the terminal screen.
- cd to change the working directory. And check for Errors.
- pwd to print the current working directory.
- history displays the last 10 commands entered by the user. commands are stored in a FIFO (First-In, First-Out) order. Each command is also paired with a number and when you do !”number” it executes the command that the number which was called was paired with.

Built in tests

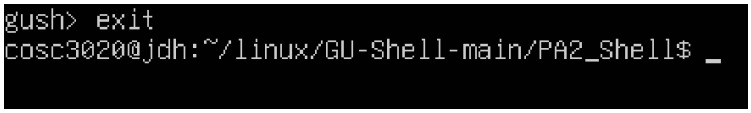
Path



History - history function and !2 command



Exit - exit



Kill pid

pwd

```
gush> pwd
/home/cosc3020/linux/GU-Shell-main/PA2_Shell
gush>
```

cd ..

```
gush> cd ..
gush> ls
PA2_Shell  README.txt
Executing command: /bin/ls
gush>
```

execute.c/h

When a user types a command in the gush shell, such as ls, the shell needs to determine where the corresponding executable file is located. This is the path resolution. In execute.c we use the PATH variable.

The PATH Variable
The shell maintains a search path called the PATH variable. This variable contains a list of directories where executables may be found. When a user types a command, the shell searches these directories in order to find the executable file. The gush shell resolves commands by checking if they are built-in or external. If external, it searches the directories in PATH, verifies executability using access(), and executes them using execve(). This process allows users to execute programs without specifying full paths each time.

gush/.c

gush is a shell that can execute commands interactively or in batch mode. It supports basic command execution, piping, input/output redirection, and built-in commands.

Gushell includes:

Interactive: Mode, shell runs in a loop, displaying the gush prompt, accepting user commands, and executing them.
Batch Mode: The shell reads commands from a file and executes them sequentially.
Process Management: Commands are executed using fork(), execve(), and wait(), ensuring proper process control.
Piping: Supports up to four pipes (| operator) to allow inter-process communication.
Built-in Commands: Includes internal commands like exit to terminate the shell.
Error Handling: Prints error messages and exits gracefully when needed.
pipes.c/h

The pipe feature in gush enhances command-line functionality by using pipes to command chain them. It supports up to 4 pipes (5 commands) and works with the redirection Process.

redirection.c/h

gush uses input (<) and output (>) redirection, allowing control where a command reads its input from or writes its output to.
In the redirection.c file, descriptors and dup2() are used to control standard input and output. Redirection needs to be the last part of a command, and only one input (<) and one output (>) redirection are allowed per command. For input redirection (<), the file must exist, or an error will occur. The output file is overwritten when using >, unless modified to append (>>).

Redirection is useful when you want to:
Save command output to a file instead of displaying it on the screen.
Use a file as input for a command instead of typing manually.

Redirection and Pipe testing

Single pipe - /bin/ps -aux | /usr/bin/nl -v 201 -i 2

```
   451  cosc3020    2721  100  0.0   8820  3968 tty1     R+   05:05   0:00 /bin/ps -aux
   453  cosc3020    2722  0.0  0.0   2276  1280 tty1     S+   05:05   0:00 /usr/bin/nl -v 201 -i 2
gush> _
```

Two or more pipes - ps -aux | grep sbin | nl

```
gush> ps -aux | grep sbin | nl
    1  root         356  0.0  0.6 290144 25984 ?        SLsl 14:18   0:03 /sbin/multipathd -d -s
    2  syslog       713  0.0  0.1 222808  4992 ?        Ssl  14:18   0:00 /usr/sbin/rsyslogd -n -iNONE
    3  root         788  0.0  0.2 398868 11392 ?        Ssl  14:18   0:00 /usr/sbin/ModemManager
    4  root         866  0.0  0.0   6748  2432 ?        Ss   14:18   0:00 /usr/sbin/cron -f -P
    5  root         873  0.0  0.0   5292  1792 ?        Ss+  14:18   0:00 /sbin/agetty -o -p -- \u --keep-baud 115200,57600,38400,9600 - vt220
    6  root        4041  0.0  0.1  12056  7040 ?        Ss   19:29   0:00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
    7  cosc3020    4375  0.0  0.0   3124  1280 tty1     S+   19:29   0:00 grep sbin
gush> _
```

Simple redirect stdin - nl -v 111 -i 200 < message.txt

```
gush> nl -v 111 -i 200 < message.txt
   111  LINE    1       Peace on earth
   311  LINE    2       Joy to the world
   511  LINE    3       hack free or die
Executing command: /bin/nl
gush>
```

Simple redirect stdout - ps -aux > tmp.txt

```
gush> ps -aux > tmp.txt
Executing command: /bin/ps
gush> _
```

Redirect stdin and stdout nl -v 111 -i 200 < message.txt > test.txt + Pipe with redirect cat < message.txt | grep 1

```
gush> nl -v 111 -i 200 < message.txt > test.txt
Executing command: /bin/nl
gush> cat < message.txt | grep 1
LINE    1       Peace on earth
```

Multiple pipes - ps -aux | grep sbin | wc -l

```
gush> ps -aux | grep sbin | wc -l
7
gush>
```

utils.c/h

The util implementation in gush, follows an error-handling protocol to ensure consistency and clarity when handling errors. Errors that are caught by gush include:

- Syntax errors (e.g., unmatched quotes, missing arguments, invalid command syntax).
- Execution errors (e.g., attempting to run a non-existent command).

Fatal Errors

Certain errors require gush to terminate immediately by calling exit(1). Examples include: When the shell is invoked with more than one batch file, the batch file provided is invalid or cannot be opened.

Sources

- https://brennan.io/2015/01/16/write-a-shell-in-c/
- https://www.geeksforgeeks.org/making-linux-shell-c/
- https://medium.com/@santiagobedoa/coding-a-shell-using-c-1ea939f10e7e
- https://stackoverflow.com/questions/28502305/writing-a-simple-shell-in-c-using-fork-execvp
- https://medium.com/@WinnieNgina/guide-to-code-a-simple-shell-in-c-bd4a3a4c41cd
- https://www.geeksforgeeks.org/linked-list-in-c/
- https://chatgpt.com/share/67bd1f81-fb10-800e-84d0-2b4e8380c7ca
- https://stackoverflow.com/questions/8082932/connecting-n-commands-with-pipes-in-a-shell
- https://cboard.cprogramming.com/c-programming/150585-implementing-pipe-my-own-unix-shell.html