



unitec[®]
Universidad Tecnológica
Centroamericana

Proyecto:

DWIMSH en Linux

Semana: 9

Nombre de los estudiante:

Víctor Isaías Romero Núñez
12211079

Sede de estudio:

UNITEC TGU

Docente:

Ing. Román Arturo Pineda Soto

Clase y Sección:

SISTEMAS OPERATIVOS I - 517

Fecha de entrega:

Viernes, 21 Marzo del 2025

INDEX

INDEX	2
1. INTRODUCCIÓN	3
2. DESCRIPCIÓN GENERAL	4
➤ CORRECCIÓN INTELIGENTE DE COMANDOS	4
➤ HISTORIAL PERSISTENTE DE COMANDOS	4
➤ AUTOCOMPLETADO MEDIANTE TABULACIÓN	5
➤ COMANDOS INTEGRADOS BÁSICOS	5
➤ INTERFAZ COLORIDA:	5
➤ PROMPT PERSONALIZADO	6
➤ MANEJO DE SEÑALES	6
3. ANÁLISIS DETALLADO	7
➤ ESTRUCTURA DEL CÓDIGO	7
1) Definiciones y estructuras de datos	7
2) Declaraciones de funciones:	8
3) Implementación de funciones:	9
4) Función principal (main):	13
4. COMPONENTES PRINCIPALES	14
➤ SISTEMA DE CARGA DE COMANDOS	14
➤ ALGORITMOS DE COMPARACIÓN	15
1) Distancia de Hamming	15
2) Distancia de Levenshtein	16
3) Detección de Anagramas	17
5. PRUEBAS Y EVALUACIÓN	18
➤ CORRECCIÓN DE ERRORES TIPOGRÁFICOS SIMPLES	18
➤ DETECCIÓN DE ANAGRAMAS	19
➤ COMANDOS CON ARGUMENTOS	19
6. CONCLUSIONES	20

1. INTRODUCCIÓN

DWIMSH (Do What I Mean Shell) es una implementación personalizada de un intérprete de comandos (shell) para sistemas Linux, desarrollado por su servidor Víctor Romero. El nombre **"DO WHAT I MEAN"** refleja el objetivo principal del proyecto: *proporcionar una experiencia de usuario mejorada mediante la corrección inteligente de errores tipográficos en comandos, ofreciendo sugerencias y alternativas cuando el usuario introduce un comando inexistente o erróneo.*

```
victor@Victor: ~/dwimsh_pro x + v
victor@Victor:~$ cd dwimsh_project/
victor@Victor:~/dwimsh_project$ gcc -o dwimsh dwimsh.c -lreadline -lm -Wall
victor@Victor:~/dwimsh_project$ ./dwimsh

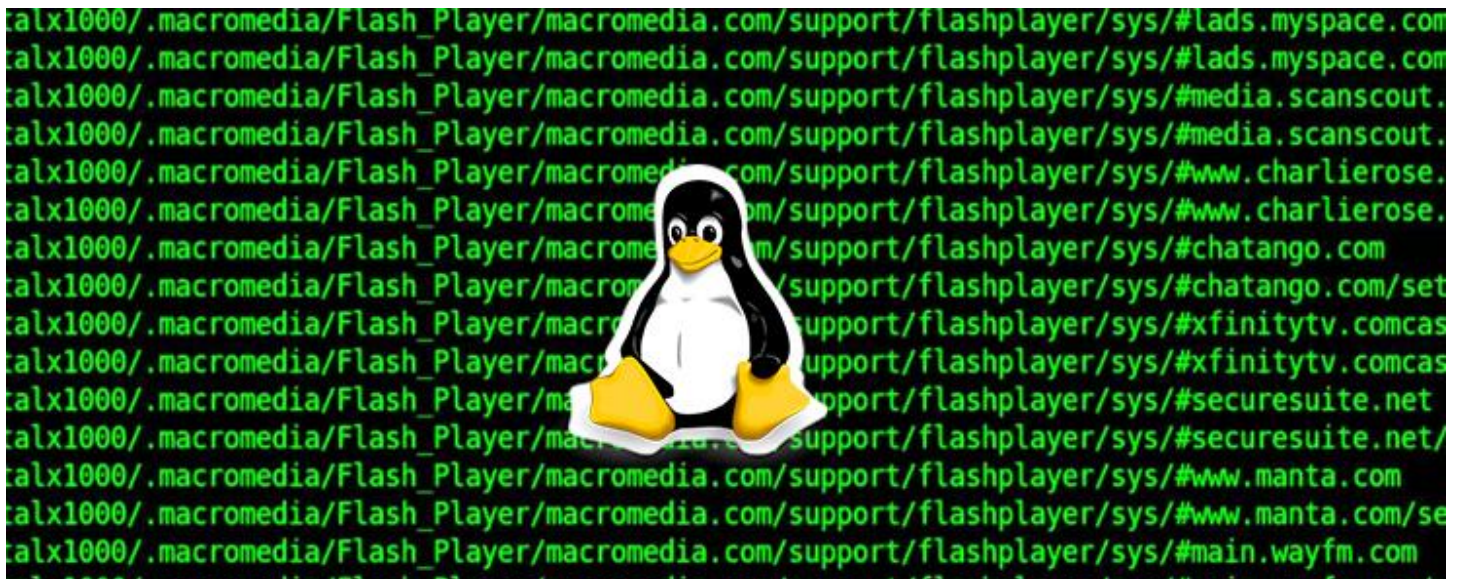
DWIMSH

Do What I Mean Shell - Linux Edition v1.0
WRITTEN BY VICTOR ROMERO - 12211079

Type 'help' for available commands or 'exit' to quit

dwimsh:~/dwimsh_project$
```

A diferencia de shells tradicionales como Bash o Zsh, DWIMSH implementa diversos algoritmos de comparación y corrección para identificar posibles comandos que el usuario intentaba ejecutar, haciendo el sistema más tolerante a errores y más intuitivo para usuarios inexpertos. Este enfoque representa un avance interesante en el campo de las interfaces de línea de comandos, centrándose en la usabilidad y la experiencia del usuario.



2. DESCRIPCIÓN GENERAL

➤ CORRECCIÓN INTELIGENTE DE COMANDOS

Utiliza múltiples algoritmos (distancia de **Hamming**, distancia de **Levenshtein** y detección de **anagramas**) para sugerir alternativas cuando se introduce un comando inexistente.

```
victor@Victor: ~/dwimsh_pro X + v
victor@Victor:~$ cd dwimsh_project/
victor@Victor:~/dwimsh_project$ gcc -o dwimsh dwimsh.c -lreadline -lm -Wall
victor@Victor:~/dwimsh_project$ ./dwimsh

DWIMSH

Do What I Mean Shell - Linux Edition v1.0
WRITTEN BY VICTOR ROMERO - 12211079

Type 'help' for available commands or 'exit' to quit

dwimsh:~/dwimsh_project$ sl
Command not found: sl
Found 7 possible commands:
Did you mean: "ls"? [y/n] y
Executing: ls
dwimsh dwimsh.c test
dwimsh:~/dwimsh_project$ |
```

➤ HISTORIAL PERSISTENTE DE COMANDOS

Almacena y recupera el historial de comandos entre sesiones en el archivo `.dwimsh_history` en el directorio home del usuario.

```
dwimsh:~/dwimsh_project$ history
 1  dc ..
 2  n
 3  help
 4  list
 5  clear
 6  exit
 7  n
 8  history
 9  exit
```

➤ AUTOCOMPLETADO MEDIANTE TABULACIÓN

Implementa la funcionalidad de completado de comandos al presionar la tecla Tab, utilizando la biblioteca readline.

```
dwimsh:~/dwimsh_project$ grep
```

➤ COMANDOS INTEGRADOS BÁSICOS

Incluye comandos internos como exit, help, clear, list y history.

```
dwimsh:~/dwimsh_project$ help
```

DWIMSH – Do What I Mean Shell

Built-in commands:

exit	- Exit the shell
help	- Display this help message
clear	- Clear the screen
list	- List all available commands
history	- Show command history

Features:

- Command correction using Hamming distance
- Command correction using Levenshtein distance
- Command correction using anagram detection
- Command history with up/down arrow keys
- Tab completion for commands

➤ INTERFAZ COLORIDA:

Utiliza códigos ANSI para mostrar texto con colores, mejorando la legibilidad y la experiencia visual.

```
victor@Victor: ~/dwimsh_pro x + v
victor@Victor:~$ cd dwimsh_project/
victor@Victor:~/dwimsh_project$ gcc -o dwimsh dwimsh.c -lreadline -lm -Wall
victor@Victor:~/dwimsh_project$ ./dwimsh
```

DWIMSH

Do What I Mean Shell – Linux Edition v1.0
WRITTEN BY VICTOR ROMERO – 12211079

Type 'help' for available commands or 'exit' to quit

```
dwimsh:~/dwimsh_project$ sl
Command not found: sl
Found 7 possible commands:
Did you mean: "ls"? [y/n] y
Executing: ls
dwimsh dwimsh.c test
dwimsh:~/dwimsh_project$ history
```

➤ PROMPT PERSONALIZADO

Muestra el directorio actual de trabajo en el prompt, reemplazando la ruta completa del directorio home por el símbolo "~".

```
dwimsh:~/dwimsh_project$ pwd  
/home/victor/dwimsh_project
```

➤ MANEJO DE SEÑALES

Implementa manejadores para señales como **SIGINT (Ctrl+C)** y **SIGTERM** para garantizar un comportamiento adecuado ante interrupciones.

```
dwimsh:~/dwimsh_project$ pwd  
/home/victor/dwimsh_project  
dwimsh:~/dwimsh_project$ mkd "Texto Copiado para prueba"
```

El programa está diseñado para funcionar en sistemas Linux y utiliza diversas bibliotecas estándar y específicas de Unix para implementar su funcionalidad.

3. ANÁLISIS DETALLADO

➤ ESTRUCTURA DEL CÓDIGO

El código fuente de DWIMSH está escrito en C y se organiza en múltiples secciones funcionales...

1) Definiciones y estructuras de datos

- Incluye múltiples bibliotecas necesarias para el funcionamiento del shell.
- Define constantes para colores ANSI, longitudes máximas y umbrales.
- Establece una estructura CommandTable para almacenar los comandos disponibles.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <sys/wait.h>
6  #include <dirent.h>
7  #include <ctype.h>
8  #include <time.h>
9  #include <signal.h>
10 #include <pwd.h>
11 #include <termios.h>
12 #include <libgen.h>
13 #include <readline/readline.h>
14 #include <readline/history.h>
15 #include <math.h>
16
17 #define MAX_CMD_LENGTH 1024
18 #define MAX_CMDS 2048
19 #define MAX_PATH_LENGTH 1024
20 #define MAX_RECOMMENDATIONS 100
21 #define HISTORY_FILE ".dwimsh_history"
22 #define LEVENSHTein_THRESHOLD 0.4 // Threshold for recommendations
23
24 // ANSI color codes
25 #define COLOR_RED "\x1b[31m"
26 #define COLOR_GREEN "\x1b[32m"
27 #define COLOR_YELLOW "\x1b[33m"
28 #define COLOR_BLUE "\x1b[34m"
29 #define COLOR_MAGENTA "\x1b[35m"
30 #define COLOR_CYAN "\x1b[36m"
31 #define COLOR_RESET "\x1b[0m"
32 #define COLOR_BOLD "\x1b[1m"
33
34 typedef struct {
35     char *commands[MAX_CMDS];
36     int count;
37 } CommandTable;
```

2) Declaraciones de funciones:

- El código sigue un enfoque de declaración anticipada de funciones.
- Las funciones están agrupadas según su propósito: inicialización, manipulación de comandos, algoritmos de comparación, etc.

```
1  CommandTable cmdTable;
2  char *homeDir;
3  char historyFilePath[MAX_PATH_LENGTH];
4  int interactive = 1;
5
6  // Function declarations
7  void LoadCommands();
8  int IsCommandInTable(const char *cmd);
9  void FreeCommandsMemory();
10 void TokenizeUserInput(char *command, char **tokens, int *tokenCount);
11 void PrintTokens(char **tokens, int tokenCount);
12 int HammingDistance(const char *str1, const char *str2);
13 int LevenshteinDistance(const char *s1, const char *s2);
14 int AreAnagrams(const char *str1, const char *str2);
15 void FindSimilarCommands(const char *cmd, char **recommendations, int *recommendationCount);
16 void JoinUserRecommendation(char *recommendation, char **tokens, int tokenCount, char *newCommand);
17 void ListCommandsTable();
18 void DeleteDuplicatedRecommendations(char **recommendations, int *recommendationCount);
19 int ExecuteCommand(char **tokens, int tokenCount);
20 int IsBuiltInCommand(const char *cmd);
21 int IsYesResponse(const char *response);
22 int IsNoResponse(const char *response);
23 void HandleSignal(int sig);
24 void InitHistory();
25 void SaveHistory();
26 void PrintWelcomeMessage();
27 void PrintHelpMessage();
28 void Cleanup();
29 char *GetPrompt();
30 void PrintColoredText(const char *text, const char *color);
31 char *CommandGenerator(const char *text, int state);
32 void InitializeCompletion();
```


3) Implementación de funciones:

- Cada función se implementa de manera modular, siguiendo principios de responsabilidad única.
- El código incluye comentarios descriptivos que explican el propósito de cada función.

```

1 // Print welcome message with ASCII art
2 void PrintWelcomeMessage() {
3     printf("\n");
4     printf(COLOR_GREEN "      _   _   _   _   _   _   _\n");
5         printf("    | |_|_|_|_|_|_|_|_|_|_|_\n");
6         printf("    | |_|_|_|_|_|_|_|_|_|_|_\n");
7         printf("    | |_|_|_|_|_|_|_|_|_|_|_\n");
8         printf("    | |_|_|_|_|_|_|_|_|_|_|_\n");
9         printf("    |_|_|_|_|_|_|_|_|_|_|_" COLOR_RESET);
10        printf(COLOR_GREEN "Do What I Mean Shell - Linux Edition v1.0\n" COLOR_RESET);
11        printf(COLOR_GREEN "WRITTEN BY VICTOR ROMERO - 12211079\n\n" COLOR_RESET);
12        printf(COLOR_YELLOW "Type 'help' for available commands or 'exit' to quit\n" COLOR_RESET);
13        printf("\n");
14 }

1 // Print help message
2 void PrintHelpMessage() {
3     printf("\n%sDWIMSH - Do What I Mean Shell%s\n\n", COLOR_BOLD, COLOR_RESET);
4     printf("Built-in commands:\n");
5     printf(" %sexit%          - Exit the shell\n", COLOR_BOLD, COLOR_RESET);
6     printf(" %shelp%           - Display this help message\n", COLOR_BOLD, COLOR_RESET);
7     printf(" %sclear%          - Clear the screen\n", COLOR_BOLD, COLOR_RESET);
8     printf(" %slist%           - List all available commands\n", COLOR_BOLD, COLOR_RESET);
9     printf(" %shistory%        - Show command history\n", COLOR_BOLD, COLOR_RESET);
10    printf("\n");
11    printf("Features:\n");
12    printf(" - Command correction using Hamming distance\n");
13    printf(" - Command correction using Levenshtein distance\n");
14    printf(" - Command correction using anagram detection\n");
15    printf(" - Command history with up/down arrow keys\n");
16    printf(" - Tab completion for commands\n");
17    printf("\n");
18 }

139 // Cleanup resources and memory
140 void Cleanup() {
141     SaveHistory();
142     FreeCommandsMemory();
143     clear_history();
144 }

145 
146 // Get current directory for prompt
147 char *GetPrompt() {
148     char cwd[MAX_PATH_LENGTH];
149     char *prompt = malloc(MAX_PATH_LENGTH);
150 
151     if (getcwd(cwd, sizeof(cwd)) == NULL) {
152         strcpy(cwd, "unknown");
153     }
154 
155     // If in home directory, use '~'
156     if (strncmp(cwd, homeDir, strlen(homeDir)) == 0) {
157         char *home_relative = cwd + strlen(homeDir);
158         if (*home_relative == '\0') {
159             sprintf(prompt, COLOR_GREEN "dwimsh" COLOR_YELLOW ":" COLOR_BLUE "~" COLOR_RESET "$ ");
160         } else {
161             sprintf(prompt, COLOR_GREEN "dwimsh" COLOR_YELLOW ":" COLOR_BLUE "%s" COLOR_RESET "$ ", home_relative);
162         }
163     } else {
164         sprintf(prompt, COLOR_GREEN "dwimsh" COLOR_YELLOW ":" COLOR_BLUE "%s" COLOR_RESET "$ ", basename(cwd));
165     }
166 
167     return prompt;
168 }

```



```
1 // Calculate the Levenshtein distance between two strings
2 int LevenshteinDistance(const char *s1, const char *s2) {
3     int len1 = strlen(s1);
4     int len2 = strlen(s2);
5
6     int matrix[len1 + 1][len2 + 1];
7
8     for (int i = 0; i <= len1; i++)
9         matrix[i][0] = i;
10
11    for (int j = 0; j <= len2; j++)
12        matrix[0][j] = j;
13
14    for (int i = 1; i <= len1; i++) {
15        for (int j = 1; j <= len2; j++) {
16            int cost = (s1[i-1] == s2[j-1]) ? 0 : 1;
17
18            int delete_cost = matrix[i-1][j] + 1;
19            int insert_cost = matrix[i][j-1] + 1;
20            int subst_cost = matrix[i-1][j-1] + cost;
21
22            int min = delete_cost;
23            if (insert_cost < min) min = insert_cost;
24            if (subst_cost < min) min = subst_cost;
25
26            matrix[i][j] = min;
27        }
28    }
29
30    return matrix[len1][len2];
31 }
```

```

1 // Find similar commands using multiple algorithms
2 void FindSimilarCommands(const char *cmd, char **recommendations, int *recommendationCount) {
3     *recommendationCount = 0;
4     int len_cmd = strlen(cmd);
5
6     if (len_cmd == 0)
7         return;
8
9     for (int i = 0; i < cmdTable.count && *recommendationCount < MAX_RECOMMENDATIONS; i++) {
10         int len_table = strlen(cmdTable.commands[i]);
11
12         // Skip very short commands
13         if (len_table < 2)
14             continue;
15
16         // Filter: skip if the difference in length is too large
17         if (fabs(len_cmd - len_table) > fmax(len_cmd, len_table) * LEVENSHTein_THRESHOLD)
18             continue;
19
20         // 1. Substring check (fast method)
21         if (strstr(cmdTable.commands[i], cmd) != NULL) {
22             recommendations[*recommendationCount] = strdup(cmdTable.commands[i]);
23             (*recommendationCount)++;
24             continue;
25         }
26
27         // 2. Hamming distance (if lengths are equal)
28         if (len_cmd == len_table) {
29             int distance = HammingDistance(cmd, cmdTable.commands[i]);
30             if (distance >= 0 && distance <= len_cmd * 0.5) {
31                 recommendations[*recommendationCount] = strdup(cmdTable.commands[i]);
32                 (*recommendationCount)++;
33                 continue;
34             }
35         }
36
37         // 3. Levenshtein distance (for different lengths)
38         int distance = LevenshteinDistance(cmd, cmdTable.commands[i]);
39         float normalized_distance = (float)distance / (float)fmax(len_cmd, len_table);
40         if (normalized_distance <= LEVENSHTein_THRESHOLD) {
41             recommendations[*recommendationCount] = strdup(cmdTable.commands[i]);
42             (*recommendationCount)++;
43             continue;
44         }
45
46         // 4. Check if they are anagrams
47         if (AreAnagrams(cmd, cmdTable.commands[i])) {
48             recommendations[*recommendationCount] = strdup(cmdTable.commands[i]);
49             (*recommendationCount)++;
50             continue;
51         }
52     }
53 }

```

```

1 // Check if two strings are anagrams
2 int AreAnagrams(const char *str1, const char *str2) {
3     if (strlen(str1) != strlen(str2))
4         return 0;
5
6     int counts[256] = {0};
7
8     for (int i = 0; i < strlen(str1); i++) {
9         counts[(unsigned char)str1[i]]++;
10        counts[(unsigned char)str2[i]]--;
11    }
12
13    for (int i = 0; i < 256; i++) {
14        if (counts[i] != 0)
15            return 0;
16    }
17    return 1;
18 }

```

```

1 // Join the recommendation with the additional arguments from the original command
2 void JoinUserRecommendation(char *recommendation, char **tokens, int tokenCount, char *newCommand) {
3     strcpy(newCommand, recommendation);
4
5     for (int i = 1; i < tokenCount; i++) {
6         strcat(newCommand, " ");
7         strcat(newCommand, tokens[i]);
8     }
9 }
10
11 // Print the table of available commands
12 void ListCommandsTable() {
13     printf("Available commands (%d total):\n", cmdTable.count);
14
15     int columns = 4;
16     int rows = (cmdTable.count + columns - 1) / columns;
17     int maxWidth = 0;
18
19     // Calculate maximum width for formatting
20     for (int i = 0; i < cmdTable.count; i++) {
21         int len = strlen(cmdTable.commands[i]);
22         if (len > maxWidth) {
23             maxWidth = len;
24         }
25     }
26
27     maxWidth += 2; // Add padding
28
29     // Print in columns
30     for (int row = 0; row < rows; row++) {
31         for (int col = 0; col < columns; col++) {
32             int index = col * rows + row;
33             if (index < cmdTable.count) {
34                 printf("%-*s", maxWidth, cmdTable.commands[index]);
35             }
36         }
37         printf("\n");
38     }
39 }
40

```

4) Función principal (main):

- Inicializa el entorno del shell.
- Implementa el bucle principal de lectura-evaluación-impresión (REPL).
- Gestiona la ejecución de comandos y el manejo de errores.

```
1 int main(int argc, char *argv[]) {
2     // Set up signal handlers
3     signal(SIGINT, HandleSignal);
4     signal(SIGTERM, HandleSignal);
5
6     // Initialize history
7     InitHistory();
8
9     // Load commands
10    LoadCommands();
11
12    // Set up readline tab completion
13    InitializeCompletion();
14
15    // Print welcome message
16    PrintWelcomeMessage();
17
18    char *input;
19    int should_exit = 0;
20
21    while (!should_exit) {
22        // Get command using readline
23        char *prompt = GetPrompt();
24        input = readline(prompt);
25        free(prompt);
26
27        // Handle EOF (Ctrl+D)
28        if (input == NULL) {
29            printf("\n");
30            break;
31        }
32
33        // Skip empty lines
34        if (input[0] == '\0') {
35            free(input);
36            continue;
37        }
38
39        // Add to history if non-empty
40        add_history(input);
41
42        // Parse the command
43        char *tokens[MAX_CMD_LENGTH];
44        int tokenCount = 0;
45
46        char *inputCopy = strdup(input);
47        TokenizeUserInput(inputCopy, tokens, &tokenCount);
48
49        if (tokenCount > 0) {
50            int result = ExecuteCommand(tokens, tokenCount);
51
52            if (result == 1) {
53                // Exit command
54                should_exit = 1;
55            } else if (result == -1) {
56                // Command not found, search for recommendations
57                printf(COLOR_RED "Command not found: %s\n" COLOR_RESET, tokens[0]);
58
59                char *recommendations[MAX_RECOMMENDATIONS];
60                int recommendationCount = 0;
61
62                FindSimilarCommands(tokens[0], recommendations, &recommendationCount);
63
64                if (recommendationCount == 0) {
65                    printf("No similar commands found. Please try again.\n");
66                } else {
67                    DeleteDuplicatedRecommendations(recommendations, &recommendationCount);
68
69                    printf(COLOR_YELLOW "Found %d possible commands:\n" COLOR_RESET,
70                        recommendationCount,
71                        recommendationCount == 1 ? "" : "s");
72
73                    char userInput[MAX_CMD_LENGTH];
74                    for (int i = 0; i < recommendationCount; i++) {
75                        printf(COLOR_CYAN "Did you mean: \"\n" COLOR_BOLD "%s", recommendations[i]);
76                        for (int j = 1; j < tokenCount; j++) {
77                            printf(" %s", tokens[j]);
78                        }
79                        printf(COLOR_RESET COLOR_CYAN "\n" COLOR_RESET);
80                    }
81
82                    if (fgets(userInput, sizeof(userInput), stdin) == NULL) {
83                        printf("\n");
84                        for (int k = 0; k < recommendationCount; k++) {
85                            free(recommendations[k]);
86                        }
87                        free(inputCopy);
88                        free(input);
89                        Cleanup();
90                        return 0;
91                    }
92
93                    userInput[strcspn(userInput, "\n")] = 0;
94
95                    if (IsValidResponse(userInput)) {
96                        char newCommand[MAX_CMD_LENGTH];
97                        char *newTokens[MAX_CMD_LENGTH];
98                        int newTokenCount = 0;
99
100                        JoinUserRecommendation(recommendations[i], tokens, tokenCount, newCommand);
101                        printf(COLOR_GREEN "Executing: %s\n" COLOR_RESET, newCommand);
102
103                        char *newCommandCopy = strdup(newCommand);
104                        TokenizeUserInput(newCommandCopy, newTokens, &newTokenCount);
105
106                        int exec_result = ExecuteCommand(newTokens, newTokenCount);
107                        if (exec_result == 1) {
108                            should_exit = 1;
109                        }
110
111                        free(newCommandCopy);
112                        break;
113                    } else if (!IsValidResponse(userInput)) {
114                        continue;
115                    } else {
116                        printf(COLOR_RED "Please enter 'y' or 'n'.\n" COLOR_RESET);
117                        i--; // Repeat the current recommendation
118                    }
119                }
120
121                for (int i = 0; i < recommendationCount; i++) {
122                    free(recommendations[i]);
123                }
124            }
125
126            free(inputCopy);
127            free(input);
128        }
129
130        Cleanup();
131
132        return 0;
133    }
134 }
```

4. COMPONENTES PRINCIPALES

➤ SISTEMA DE CARGA DE COMANDOS


La función LoadCommands() es responsable de cargar todos los comandos ejecutables disponibles en las rutas especificadas en la variable de entorno PATH, Este componente asegura que DWIMSH tenga acceso a todos los comandos disponibles en el sistema, los cuales serán utilizados posteriormente para las sugerencias de corrección.

```
1 // Load commands from PATH directories
2 void LoadCommands() {
3     char *path = getenv("PATH");
4     char *pathCopy = strdup(path);
5     char *dir = strtok(pathCopy, ":");
6     DIR *dirp;
7     struct dirent *entry;
8
9     cmdTable.count = 0;
10
11     while (dir != NULL && cmdTable.count < MAX_CMDS) {
12         dirp = opendir(dir);
13         if (dirp != NULL) {
14             while ((entry = readdir(dirp)) != NULL && cmdTable.count < MAX_CMDS) {
15                 if (entry->d_type == DT_REG || entry->d_type == DT_LNK) {
16                     // Check if the file is executable
17                     char fullPath[MAX_PATH_LENGTH];
18                     snprintf(fullPath, MAX_PATH_LENGTH, "%s/%s", dir, entry->d_name);
19                     if (access(fullPath, X_OK) == 0) {
20                         cmdTable.commands[cmdTable.count] = strdup(entry->d_name);
21                         cmdTable.count++;
22                     }
23                 }
24             }
25             closedir(dirp);
26         }
27         dir = strtok(NULL, ":");
28     }
29
30     free(pathCopy);
31
32     // Sort commands alphabetically for easier searching
33     for (int i = 0; i < cmdTable.count; i++) {
34         for (int j = i + 1; j < cmdTable.count; j++) {
35             if (strcmp(cmdTable.commands[i], cmdTable.commands[j]) > 0) {
36                 char *temp = cmdTable.commands[i];
37                 cmdTable.commands[i] = cmdTable.commands[j];
38                 cmdTable.commands[j] = temp;
39             }
40         }
41     }
42
43     // Add built-in commands
44     if (cmdTable.count < MAX_CMDS) {
45         cmdTable.commands[cmdTable.count++] = strdup("exit");
46     }
47     if (cmdTable.count < MAX_CMDS) {
48         cmdTable.commands[cmdTable.count++] = strdup("help");
49     }
50     if (cmdTable.count < MAX_CMDS) {
51         cmdTable.commands[cmdTable.count++] = strdup("clear");
52     }
53     if (cmdTable.count < MAX_CMDS) {
54         cmdTable.commands[cmdTable.count++] = strdup("list");
55     }
56     if (cmdTable.count < MAX_CMDS) {
57         cmdTable.commands[cmdTable.count++] = strdup("history");
58     }
59 }
```

➤ ALGORITMOS DE COMPARACIÓN

DWIMSH implementa tres algoritmos principales para encontrar comandos similares:

1) Distancia de Hamming



```
1 // Calculate the Hamming distance between two strings
2 int HammingDistance(const char *str1, const char *str2) {
3     if (strlen(str1) != strlen(str2))
4         return -1;
5
6     int distance = 0;
7     for (int i = 0; i < strlen(str1); i++) {
8         if (str1[i] != str2[i]) {
9             distance++;
10        }
11    }
12    return distance;
13 }
```

2) Distancia de Levenshtein

El algoritmo de distancia de Levenshtein implementado en DWIMSH calcula la "distancia de edición" entre dos cadenas, es decir, el número mínimo de operaciones (inserción, eliminación o sustitución) necesarias para transformar una cadena en otra.

```
1 // Calculate the Levenshtein distance between two strings
2 int LevenshteinDistance(const char *s1, const char *s2) {
3     int len1 = strlen(s1);
4     int len2 = strlen(s2);
5
6     int matrix[len1 + 1][len2 + 1];
7
8     for (int i = 0; i <= len1; i++)
9         matrix[i][0] = i;
10
11    for (int j = 0; j <= len2; j++)
12        matrix[0][j] = j;
13
14    for (int i = 1; i <= len1; i++) {
15        for (int j = 1; j <= len2; j++) {
16            int cost = (s1[i-1] == s2[j-1]) ? 0 : 1;
17
18            int delete_cost = matrix[i-1][j] + 1;
19            int insert_cost = matrix[i][j-1] + 1;
20            int subst_cost = matrix[i-1][j-1] + cost;
21
22            int min = delete_cost;
23            if (insert_cost < min) min = insert_cost;
24            if (subst_cost < min) min = subst_cost;
25
26            matrix[i][j] = min;
27        }
28    }
29
30    return matrix[len1][len2];
31 }
```


3) Detección de Anagramas



```
1 // Check if two strings are anagrams
2 int AreAnagrams(const char *str1, const char *str2) {
3     if (strlen(str1) != strlen(str2))
4         return 0;
5
6     int counts[256] = {0};
7
8     for (int i = 0; i < strlen(str1); i++) {
9         counts[(unsigned char)str1[i]]++;
10        counts[(unsigned char)str2[i]]--;
11    }
12
13    for (int i = 0; i < 256; i++) {
14        if (counts[i] != 0)
15            return 0;
16    }
17    return 1;
18 }
19
```

5. PRUEBAS Y EVALUACIÓN

➤ CORRECCIÓN DE ERRORES TIPOGRÁFICOS SIMPLES

1) Comando introducido: ls

Comportamiento: El comando se ejecuta correctamente, mostrando el contenido del directorio actual.

```
victor@Victor:~/dwimsh_project$ ./dwimsh
DWIMSH
Do What I Mean Shell - Linux Edition v1.0
WRITTEN BY VICTOR ROMERO - 12211079

Type 'help' for available commands or 'exit' to quit

dwimsh:~/dwimsh_project$ ls
dwimsh  dwimsh.c  test
```

2) Comando introducido: pit

Comportamiento esperado: DWIMSH detecta que "pit" no es un comando válido y sugiere alternativas como "git".

```
dwimsh:~/dwimsh_project$ pit
Command not found: pit
Found 2 possible commands:
Did you mean: "git"? [y/n] y
Executing: git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
        [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
        [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
        [--super-prefix=<path>] [--config-env=<name>=<envvar>]
        <command> [<args>]
```

➤ DETECCIÓN DE ANAGRAMAS

1) Comando introducido: pign

Comportamiento esperado: DWIMSH detecta que "pign" es un anagrama de "ping"

```
dwimsh:~/dwimsh_project$ pign
Command not found: pign
Found 3 possible commands:
Did you mean: "pico"? [y/n] n
Did you mean: "ping"? [y/n] y
Executing: ping
```

1) Comando introducido: grpe

Comportamiento esperado: DWIMSH detecta que "grpe" es un anagrama de "grep"

```
dwimsh:~/dwimsh_project$ grpe
Command not found: grpe
Found 5 possible commands:
Did you mean: "arpd"? [y/n] n
Did you mean: "free"? [y/n] n
Did you mean: "grep"? [y/n] y
Executing: grep
Usage: grep [OPTION]... PATTERNS [FILE]...
```

➤ COMANDOS CON ARGUMENTOS

1) Comando introducido: sl -la

Comportamiento esperado: DWIMSH detecta que "sl" no es un comando válido y sugiere "ls".

```
dwimsh:~/dwimsh_project$ sl -la
Command not found: sl
Found 7 possible commands:
Did you mean: "ls -la"? [y/n] y
Executing: ls -la
total 72
drwxr-xr-x  3 victor victor  4096 Mar 21 11:58 .
drwxr-x--- 14 victor victor  4096 Mar 19 23:28 ..
-rwxr-xr-x  1 victor victor 35640 Mar 21 11:58 dwimsh
-rw-r--r--  1 victor victor 23645 Mar 18 23:23 dwimsh.c
drwxr-xr-x  2 victor victor  4096 Mar 18 23:27 test
```

2) Comando introducido: mkdri Test

Comportamiento esperado: DWIMSH detecta que "mkdri" no es un comando válido y sugiere "mkdir".

```
dwimsh:~/dwimsh_project$ mkdri Test
Command not found: mkdri
Found 1 possible command:
Did you mean: "mkdir Test"? [y/n] y
Executing: mkdir Test
dwimsh:~/dwimsh_project$ ls
Test  dwimsh  dwimsh.c  test
```

6. CONCLUSIONES

DWIMSH representa un enfoque innovador en el diseño de interfaces de línea de comandos, priorizando la usabilidad y la tolerancia a errores. Su implementación de múltiples algoritmos de corrección y sugerencia demuestra un esfuerzo por hacer que la experiencia de usuario en la línea de comandos sea más accesible, especialmente para usuarios menos experimentados.

Las principales fortalezas del proyecto incluyen:

1. **Diseño centrado en el usuario:** La corrección inteligente de comandos y las sugerencias interactivas facilitan el uso del shell, reduciendo la frustración asociada con errores tipográficos.
2. **Implementación técnica sólida:** El uso de algoritmos conocidos como Levenshtein y Hamming, combinados con detección de anagramas, proporciona un sistema de sugerencias bastante efectivo.
3. **Características modernas:** La integración con readline para historial y autocompletado, junto con una interfaz colorida, mejoran significativamente la experiencia del usuario.

Tomando en cuenta lo anterior, DWIMSH representa un punto de partida prometedor para el desarrollo de shells más intuitivos y orientados al usuario. Con mejoras adicionales en rendimiento, seguridad y funcionalidad, podría convertirse en una alternativa viable para usuarios que priorizan la facilidad de uso sobre la potencia y complejidad de shells tradicionales.

https://github.com/VIRN2005/ProyectoOps1_DWIMSH

The screenshot shows the GitHub repository page for 'ProyectoOps1_DWIMSH' by user 'VIRN2005'. The repository is public and has 0 stars and 0 forks. The main branch is 'main'. The repository contains files: README.md, dwimsh, and dwimsh.c. The README.md file is selected and displays the following content:

DWIMSH - Do What I Mean Shell

Descripción

DWIMSH es un shell interactivo para Linux que ayuda a los usuarios a corregir errores tipográficos en comandos mediante:

- Distancia de Hamming
- Distancia de Levenshtein
- Detección de anagramas
- Recomendaciones basadas en historial

Incluye funcionalidades como historial de comandos, autocompletado con `readline`, colores ANSI y manejo de señales.

Características principales

- Corrección de errores tipográficos en comandos mal escritos.
- Historial de comandos con navegación mediante flechas.
- Sugerencias inteligentes de comandos similares.
- Soporte para señales (`SIGINT`, `SIGTERM`) para manejo seguro.

The right sidebar shows repository statistics: 0 stars, 1 watching, 0 forks. It also includes sections for Releases, Packages, Languages (C 100.0%), and Suggested workflows (SLSA Generic generator).