

# File Explorer Application (C++, Linux Terminal-Based)

## Description

A console-based file explorer is a command-line program written in C++ that lets users manage their files and directories without a graphical interface. It directly interacts with the Linux operating system APIs to provide essential file operations, navigation, and permissions management – all within the terminal.

## Objective

- **Main Goal:**  
Build a terminal application in C++ that can list, search, and manipulate files and directories, offering a user-friendly console alternative to graphical file browsers.

## Core Features

- **File Listing:**  
Display contents (files and directories) of the current working directory.
- **Directory Navigation:**  
Move into subdirectories, go back to parent, or jump to specific paths.
- **File Operations:**
  - **Copy:** Duplicate files or folders.
  - **Move:** Relocate items to other directories.
  - **Delete:** Remove files or directories with safety checks.
  - **Create:** Make new files or directories.
- **File Search:**  
Search for files or folders by name or attribute.
- **File Permissions Management:**  
View and modify read, write, execute permissions for files/directories.

## Workflow / Implementation Steps

Step	Details
Design Structure	Plan core commands, user interface, error handling, and modular code structure
Set Up Coding Env	Install Linux (or use WSL on Windows), g++ compiler, text editor (VSCode, Vim, etc.)
Directory Listing	Use Linux system calls (opendir, readdir, stat) via C++ to display contents
Navigation	Allow users to change current directory (chdir, path management)
File Operations	Implement actions for copying, moving, deleting (remove), and creating files direct in C++
Search	Develop search by name or pattern within directories
Permission Ctrl	Check and modify file/directory permissions (chmod, stat)
Testing	Run the explorer, handle errors, improve usability

---

**Code:**

```
C++ terminal_explorer.cpp X
C++ terminal_explorer.cpp > ...
1  #include <iostream>
2  #include <filesystem>
3  #include <vector>
4  #include <string>
5  #include <iomanip>
6  #include <cstring>
7  #include <fstream>
8  #include <algorithm>
9  #include <chrono>
10 #include <ctime>
11
12 #ifdef _WIN32
13 #include <windows.h>
14 #include <lmcons.h>
15 #else
16 #include <unistd.h>
17 #include <sys/stat.h>
18 #include <sys/types.h>
19 #include <pwd.h>
20 #include <grp.h>
21 #endif
22
23 namespace fs = std::filesystem;
24
25 class FileExplorer {
26 private:
27     fs::path currentPath;
28     std::vector<fs::directory_entry> currentEntries;
29
30     void listDirectory() {
31         currentEntries.clear();
32         for (const auto& entry : fs::directory_iterator(currentPath)) {
33             currentEntries.push_back(entry);
34         }
35     }
36
37     void printHeader() {
38         std::cout << "File Explorer\n";
39     }
40
41     void printCurrentPath() {
42         std::cout << "Current Path: " << currentPath << "\n";
43     }
44
45     void printCurrentEntries() {
46         std::cout << "Current Entries:\n";
47         for (const auto& entry : currentEntries) {
48             std::cout << entry.path() << "  ";
49             if (entry.is_directory()) {
50                 std::cout << "[DIR]";
51             } else {
52                 std::cout << "[FILE]";
53             }
54             std::cout << "\n";
55         }
56     }
57
58     void printFooter() {
59         std::cout << "Press Ctrl+C to exit.\n";
60     }
61
62     void run() {
63         printHeader();
64         printCurrentPath();
65         listDirectory();
66         printCurrentEntries();
67         printFooter();
68     }
69
70 public:
71     FileExplorer() {
72         run();
73     }
74 };
75
76 int main() {
77     FileExplorer explorer;
78     return 0;
79 }
```

```

130     }
131 }
132
133 void createDirectory(const std::string& name) {
134     try {
135         fs::create_directory(currentPath / name);
136         listDirectory();
137     } catch (const std::exception& e) {
138         std::cerr << "Error creating directory: " << e.what() << std::endl;
139     }
140 }
141
142 void removeFile(const std::string& name) {
143     try {
144         fs::path target = currentPath / name;
145         if (fs::is_directory(target)) {
146             fs::remove_all(target);
147         } else {
148             fs::remove(target);
149         }
150         listDirectory();
151     } catch (const std::exception& e) {
152         std::cerr << "Error removing: " << e.what() << std::endl;
153     }
154 }
155
156 void copyFile(const std::string& src, const std::string& dest) {
157     try {
158         fs::path sourcePath = (src[0] == '/') ? src : currentPath / src;
159         fs::path destPath = (dest[0] == '/') ? dest : currentPath / dest;
160
161         if (fs::is_directory(sourcePath)) {

```

```

161         if (fs::is_directory(sourcePath)) {
162             fs::create_directories(destPath);
163             fs::copy(sourcePath, destPath, fs::copy_options::recursive);
164         } else {
165             fs::copy(sourcePath, destPath, fs::copy_options::overwrite_existing);
166         }
167         listDirectory();
168     } catch (const std::exception& e) {
169         std::cerr << "Error copying: " << e.what() << std::endl;
170     }
171 }
172
173 void moveFile(const std::string& src, const std::string& dest) {
174     try {
175         fs::path sourcePath = (src[0] == '/') ? src : currentPath / src;
176         fs::path destPath = (dest[0] == '/') ? dest : currentPath / dest;
177
178         fs::rename(sourcePath, destPath);
179         listDirectory();
180     } catch (const std::exception& e) {
181         std::cerr << "Error moving: " << e.what() << std::endl;
182     }
183 }
184
185 void search(const std::string& pattern) {
186     std::cout << "Searching for: " << pattern << " in " << currentPath << "\n";
187     size_t count = 0;
188
189     for (const auto& entry : fs::recursive_directory_iterator(currentPath)) {
190         std::string filename = entry.path().filename().string();
191         if (filename.find(pattern) != std::string::npos) {
192             std::cout << entry.path().string() << std::endl;

```

```

193         count++;
194     }
195 }
196
197     std::cout << "\nFound " << count << " matches.\n";
198 }
199 };
200
201 int main() {
202     FileExplorer explorer;
203     std::string command;
204
205     while (true) {
206         explorer.list();
207
208         std::cout << "\n> ";
209         std::getline(std::cin, command);
210
211         if (command == "exit") {
212             break;
213         } else if (command.substr(0, 3) == "cd ") {
214             explorer.changeDirectory(command.substr(3));
215         } else if (command.substr(0, 6) == "mkdir ") {
216             explorer.createDirectory(command.substr(6));
217         } else if (command.substr(0, 3) == "rm ") {
218             explorer.removeFile(command.substr(3));
219         } else if (command.substr(0, 3) == "cp ") {
220             size_t space = command.find(' ', 3);
221             if (space != std::string::npos) {
222                 std::string src = command.substr(3, space - 3);
223                 std::string dest = command.substr(space + 1);
224                 explorer.copyFile(src, dest);

```

```

98     std::string color = is_directory(entry) ? "\033[1;34m" : "\033[0m";
99     std::cout << " " << color << entry.path().filename().string() << "\033[0m" << std::endl;
100 }
101
102 public:
103     FileExplorer() {
104         currentPath = fs::current_path();
105         listDirectory();
106     }
107
108     void list() {
109 #ifdef _WIN32
110         system("cls");
111 #else
112         system("clear");
113 #endif
114         std::cout << "Current directory: " << currentPath << "\n\n";
115
116         for (const auto& entry : currentEntries) {
117             printFileInfo(entry);
118         }
119
120         std::cout << "\nCommands: cd <dir>, mkdir <name>, rm <name>, cp <src> <dest>, mv <src> <dest>";
121     }
122
123     void changeDirectory(const std::string& path) {
124         fs::path newPath = currentPath / path;
125         if (fs::exists(newPath) && fs::is_directory(newPath)) {
126             currentPath = fs::canonical(newPath);
127             listDirectory();
128         } else {
129             std::cerr << "Directory not found: " << path << std::endl;

```

```

198         std::string color = is_directory(entry) ? "\033[1;34m" : "\033[0m";
199         std::cout << " " << color << entry.path().filename().string() << "\033[0m" << std::endl;
200     }
201
202     public:
203     FileExplorer() {
204         currentPath = fs::current_path();
205         listDirectory();
206     }
207
208     void list() {
209     #ifdef _WIN32
210         system("cls");
211     #else
212         system("clear");
213     #endif
214         std::cout << "Current directory: " << currentPath << "\n\n";
215
216         for (const auto& entry : currentEntries) {
217             printFileInfo(entry);
218         }
219
220         std::cout << "\nCommands: cd <dir>, mkdir <name>, rm <name>, cp <src> <dest>, mv <src> <dest>" << std::endl;
221     }
222
223     void changeDirectory(const std::string& path) {
224         fs::path newPath = currentPath / path;
225         if (fs::exists(newPath) && fs::is_directory(newPath)) {
226             currentPath = fs::canonical(newPath);
227             listDirectory();
228         } else {
229             std::cerr << "Directory not found: " << path << std::endl;

```



```

226     } else if (command.substr(0, 3) == "mv ") {
227         size_t space = command.find(' ', 3);
228         if (space != std::string::npos) {
229             std::string src = command.substr(3, space - 3);
230             std::string dest = command.substr(space + 1);
231             explorer.moveFile(src, dest);
232         }
233     } else if (command.substr(0, 7) == "search ") {
234         explorer.search(command.substr(7));
235         std::cout << "Press Enter to continue...";
236         std::cin.ignore();
237     }
238 }
239
240 return 0;
241 }
242

```

## Output

```

-rwxrwxrwx user group 3912237 C-A-P-Y-B-A-R-A.mp3
-rwxrwxrwx user group 93769 COA_LAB-4.pdf
-rwxrwxrwx user group 93769 COA_LAB-5.pdf
-rwxrwxrwx user group 5222586 CS345-Algorithms-II-Algorithm-Design-by-Jon-Kleinberg-Eva-Tardos.pdf
-rwxrwxrwx user group 1668254 Campus Process.pdf
-rwxrwxrwx user group 4920544 Can you handle pressure.mp4
-rwxrwxrwx user group 4029741 Cat [Youtube_ Feline Music].mp3
-rwxrwxrwx user group 517494 Ch-21.pdf
-rwxrwxrwx user group 610981 Ch-22.pdf
-rwxrwxrwx user group 941545 Ch-24.pdf
-rwxrwxrwx user group 946040 Ch-25.pdf
-rwxrwxrwx user group 567922 Ch-26.pdf
-rwxrwxrwx user group 1387388 Chapter 2 - thinking like an economist.PDF
-rwxrwxrwx user group 489 ChatGPT Image Apr 6, 2025, 09_45_24 AM.png
-rwxrwxrwx user group 1104440 ChatGPT Installer.exe
-rwxrwxrwx user group 1373744 ChromeSetup.exe
-rwxrwxrwx user group 321600 Course Outcomes and lesson plan (PME) sem-2.pdf
-rwxrwxrwx user group 119549160 CursorUserSetup-x64-1.5.7.exe
-rwxrwxrwx user group 18038784 DB.Browser.for.SQLite-3.12.2-win64.msi
-rwxrwxrwx user group 1832038 DBMS Assignment-output.pdf
-rwxrwxrwx user group 334433 DSA Assignment-2-1 (1).pdf
-rwxrwxrwx user group 334433 DSA Assignment-2-1 (2).pdf
-rwxrwxrwx user group 334433 DSA Assignment-2-1.pdf
-rwxrwxrwx user group 0 DT20245990226.Application_Form.pdf
-rwxrwxrwx user group 9055819 DashingImage.jpg
-rwxrwxrwx user group 732396 Default_a_scenary_of_mountains_and_rising_sun_reddish_yellowda_2_651de51f-3005-4366-be0a-e6c2e888156a_1 (1).jpg
-rwxrwxrwx user group 732396 Default_a_scenary_of_mountains_and_rising_sun_reddish_yellowda_2_651de51f-3005-4366-be0a-e6c2e888156a_1 (2).jpg
-rwxrwxrwx user group 732396 Default_a_scenary_of_mountains_and_rising_sun_reddish_yellowda_2_651de51f-3005-4366-be0a-e6c2e888156a_1.jpg
-rwxrwxrwx user group 4734 Document (4) (3).pdf
-rwxrwxrwx user group 407312 DreamShaper_v5_a_creapy_mummy_frontview_angle_highly_detaile_3.jpg
-rwxrwxrwx user group 373140 DreamShaper_v5_a_warrior_face_strong_looking_straight_to_fron_2 (1).jpg
-rwxrwxrwx user group 633548 DreamShaper_v5_a_warrior_face_strong_looking_straight_to_fron_2 (2).jpg
-rwxrwxrwx user group 351287 DreamShaper_v5_a_warrior_face_strong_looking_straight_to_fron_2.jpg
-rwxrwxrwx user group 462421 DreamShaper_v5_centered_isometric_mural_graffiti_composition_s_2.jpg
-rwxrwxrwx user group 453987 DreamShaper_v5_centered_isometric_mural_graffiti_composition_s_3.jpg
-rwxrwxrwx user group 391044 DreamShaper_v5_create_image_of_a_man_with_Indian_tradational_c_3.jpg
-rwxrwxrwx user group 520620 DreamShaper_v5_dog_is_jumping_in_eyes_0.jpg
-rwxrwxrwx user group 551484 DreamShaper_v5_dog_is_jumping_in_eyes_1.jpg
-rwxrwxrwx user group 524674 DreamShaper_v5_make_the_selected_image_more_charismatic_1.jpg
-rwxrwxrwx user group 564285 DreamShaper_v5_shah_rukh_khan_2.jpg
-rwxrwxrwx user group 425074 DreamShaper_v5_shah_rukh_khan_and_priyanka_chopra_0.jpg

```

## Conclusion

The File Explorer Application in C++ for the Linux terminal enables users to efficiently manage files and directories without relying on graphical tools. By implementing core commands like listing contents, navigation, file operations, search, and permission management, you develop a solid grasp of Linux system calls and low-level programming concepts. This project offers hands-on experience with OS-level tasks, making it a practical exercise for building foundational skills in systems programming with C++.

## Future Enhancements

- User Interface Improvements: Add support for colored output, tree-like directory views, and interactive menus to enhance user experience.
- File Preview Features: Integrate file type detection and previews for text and image files.
- Batch Operations: Enable bulk file copying, moving, and deletion via pattern matching or selection.
- Bookmarking & History: Implement directory bookmarking and history tracking for quick navigation.
- Remote Access: Extend file operations to networked or remote directories using protocols like SFTP or SSH.
- Multi-threading: Make certain file operations concurrent for better performance on large datasets.

## References

- Linux Programming with C++
- man pages for opendir, readdir, stat, chmod, unlink
- Official GNU C++ Library documentation
- TutorialsPoint: File Handling in C++
- GeeksforGeeks: Directory Handling in C++