File Explorer Application (LinuxOS)

Submitted By: Khushi Kumari
Roll Number / ID: 2241011222
Course : Capstone Project(LinuxOS)
Date of Submission: 09 November 2025

1.Abstract

This project focuses on developing a **console-based File Explorer** using **C++ on Linux (Ubuntu / WSL)**.
The primary objective is to simulate the functionalities of a traditional graphical file explorer within a command-line environment, allowing users to perform file management operations such as **viewing, creating, deleting, copying, moving, and searching files or directories**.

The project enhances understanding of **Linux system programming**, **file handling**, and **permission management**. It also provides a foundation for learning how operating systems interact with files and directories at a system level.

2. Objectives

- To design a **command-line-based File Explorer** using C++ on Linux.
- To learn **system-level file operations** through Linux system calls.
- To manage files and directories using commands like create, copy, move, delete, and search.
- To understand and manipulate **file permissions** in a Linux environment.
- To gain hands-on experience with **version control (Git/GitHub)** and software documentation.

3. Technologies and Tools Used

Tool / Technology	Purpose
C++ (g++)	Implementation of the File Explorer logic
Linux / Ubuntu (WSL 2)	Execution and testing environment
Git & GitHub	Source code management and version control
Visual Studio Code	Development and editing environment
Bash Terminal	Command-line interface for testing
System Calls: `opendir`, `readdir`, `mkdir`, `chmod`, `stat`, `rename`	Core Linux functions for file operations

4.Implementation Plan

Day 1 – Setup and Basic File Listing

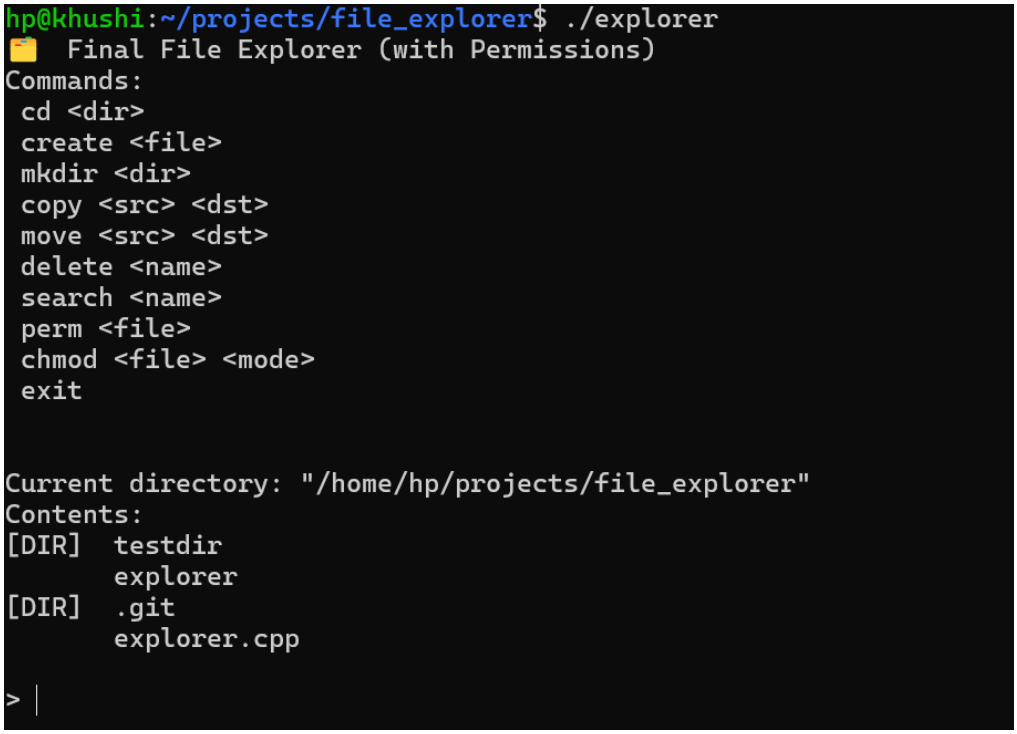
Description:

- Installed WSL 2 and Ubuntu.
- Created project folder `file_explorer`.
- Wrote and compiled the first C++ program to list directory files.

Source code:

```
#include <iostream>

#include <filesystem>

namespace fs = std::filesystem;

int main() {

    fs::path current = fs::current_path();

    std::cout << "Listing files in: " << current << "\n\n";

    for (const auto &entry : fs::directory_iterator(current)) {

        std::cout << (entry.is_directory() ? "[DIR] " : "      ")

                  << entry.path().filename().string() << "\n";

    }   return 0;

}
```

```
sudo apt install wsl
hp@khushi:~/projects/file_explorer$ lsb_release -a
uname -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 24.04.3 LTS
Release:        24.04
Codename:       noble
Linux khushi 6.6.87.2-microsoft-standard-WSL2 #1 SMP PREEMPT_DYN
AMIC Thu Jun  5 18:30:46 UTC 2025 x86_64 x86_64 x86_64 GNU/Linux
hp@khushi:~/projects/file_explorer$
```



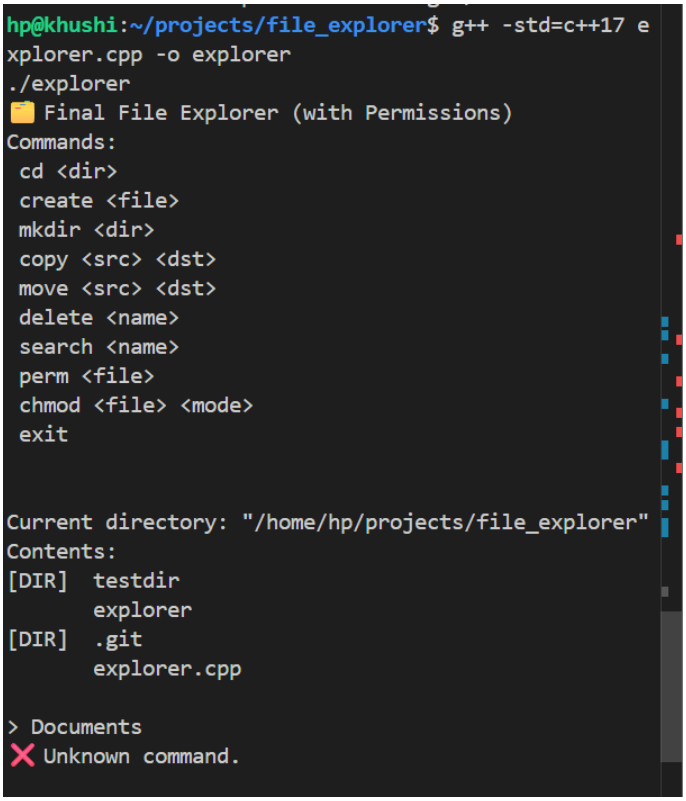
Day 2 – Directory Navigation

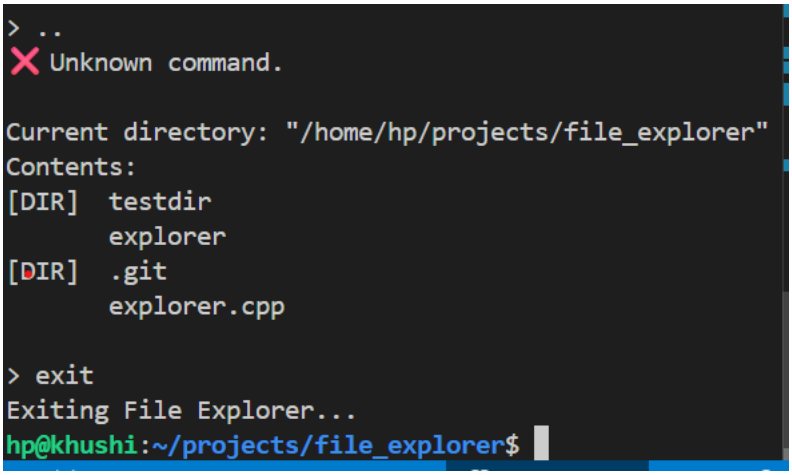
- Added navigation commands:
  - `cd` → Change directory
  - `ls` → List files

source code:

```
#include <iostream>

#include <filesystem>

#include <string>

namespace fs = std::filesystem;

int main() {

    fs::path current = fs::current_path();  // Start in current directory

   std::cout << "📂 Simple File Explorer\n";

    std::cout << "Type a folder name to enter it, '..' to go up, or 'exit' to quit.\n\n";

while (true) {

        // Show where we are
```

```
    std::cout << "Current directory: " << current << "\n";
     std::cout << "Contents:\n";
       // List files and directories
       for (const auto &entry : fs::directory_iterator(current)) {
           std::cout << (entry.is_directory() ? "[DIR]  " : "       ");
           std::cout << entry.path().filename().string() << "\n";
       }
       // Ask user what to do
       std::cout << "\nEnter directory name (or '..' / exit): ";
       std::string choice;
       std::getline(std::cin, choice);
       if (choice == "exit")
           break;
       else if (choice == "..") {
           if (current.has_parent_path())
               current = current.parent_path();
       } else {
           fs::path newPath = current / choice;
           if (fs::is_directory(newPath))
               current = newPath;
           else
               std::cout << "❌ Not a directory.\n";
       }
       std::cout << "\n";
   }
   std::cout << "Exiting File Explorer...\n";
   return 0;
}
```





Day 3 – File Manipulation

- Implemented file operations:
  - `create <filename>` → Create a new file
  - `copy <src> <dest>` → Copy files
  - `move <src> <dest>` → Move or rename files
  - `delete <filename>` → Delete files

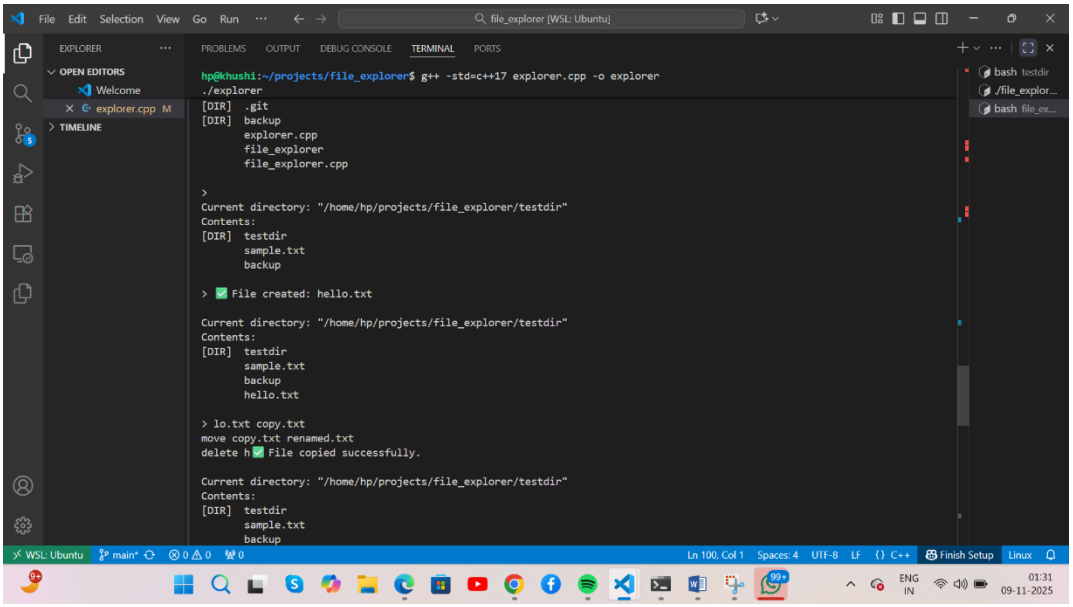
source code:

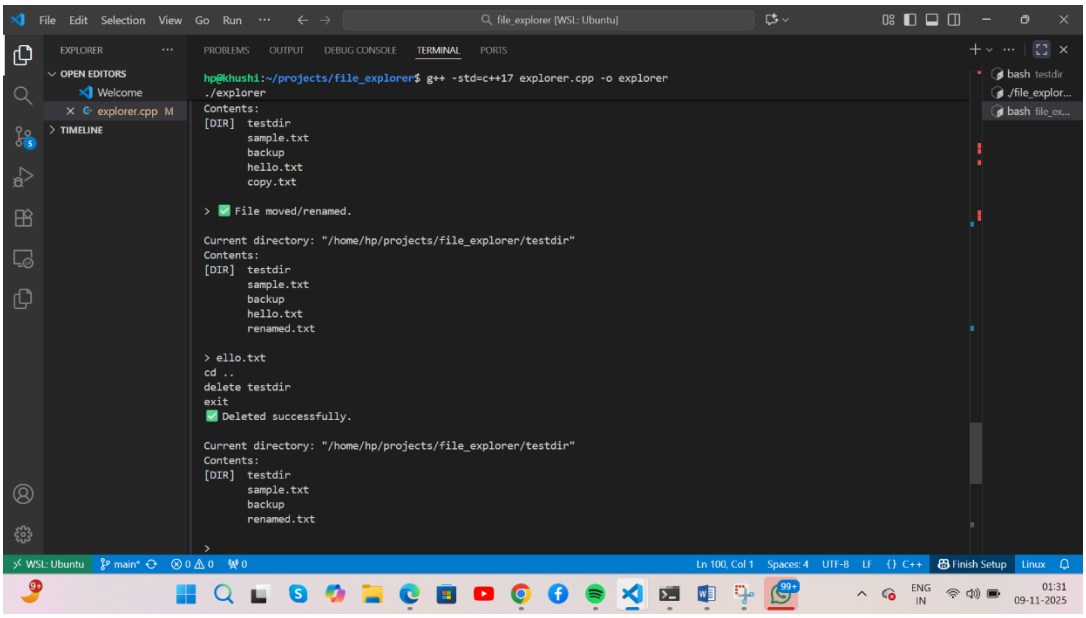
```
#include <iostream>

#include <filesystem>
```

```
#include <string>

#include <fstream>

namespace fs = std::filesystem;

void listDirectory(const fs::path& current) {

    std::cout << "\nCurrent directory: " << current << "\n";

    std::cout << "Contents:\n";

    for (const auto& entry : fs::directory_iterator(current)) {

        std::cout << (entry.is_directory() ? "[DIR]  " : "       ");

        std::cout << entry.path().filename().string() << "\n";

    }}

int main() {

    fs::path current = fs::current_path();

    std::cout << "  File Explorer with File Operations\n";

    std::cout << "Commands: cd <dir>, create <file>, mkdir <dir>, copy <src> <dst>, move <src> <dst>, delete <name>, exit\n\n";

    while (true) {

        listDirectory(current);

        std::cout << "\n> ";

        std::string command;

        std::cin >> command;

        if (command == "exit") break;

        else if (command == "cd") {

            std::string dir;

            std::cin >> dir;

            if (dir == ".." && current.has_parent_path()) {

                current = current.parent_path();

            } else if (fs::is_directory(current / dir)) {

                current /= dir;

            } else {
```

```
std::cout << "❌ Directory not found.\n";
            }
        }
        else if (command == "create") {
            std::string filename;
            std::cin >> filename;
            std::ofstream file(current / filename);
            if (file) std::cout << "✅ File created: " << filename << "\n";
            else std::cout << "❌ Failed to create file.\n";
        }
        else if (command == "mkdir") {
            std::string dirname;
            std::cin >> dirname;
            if (fs::create_directory(current / dirname))
                std::cout << "✅ Directory created: " << dirname << "\n";
            else
                std::cout << "❌ Could not create directory.\n";
        }
    else if (command == "copy") {
            std::string src, dst;
            std::cin >> src >> dst;
            try {
                fs::copy_file(current / src, current / dst, fs::copy_options::overwrite_existing);
                std::cout << "✅ File copied successfully.\n";
            } catch (...) {
                std::cout << "❌ Copy failed.\n";
            }
        }
```

```
        else if (command == "move") {
            std::string src, dst;
            std::cin >> src >> dst;
            try {
                fs::rename(current / src, current / dst);
                std::cout << "✅ File moved/renamed.\n";
            } catch (...) {
                std::cout << "❌ Move failed.\n";
            }
        }
        else if (command == "delete") {
            std::string target;
            std::cin >> target;
            try {
                fs::remove_all(current / target);
                std::cout << "✅ Deleted successfully.\n";
            } catch (...) {
                std::cout << "❌ Delete failed.\n";
            }
        }
        else {
            std::cout << "❌ Unknown command.\n";
        }
    }
    std::cout << "Exiting File Explorer...\n";
    return 0;
}
```





Day 4 – File Search

- Added recursive file search using directory traversal.
- Displays file paths when names match.

Source code:

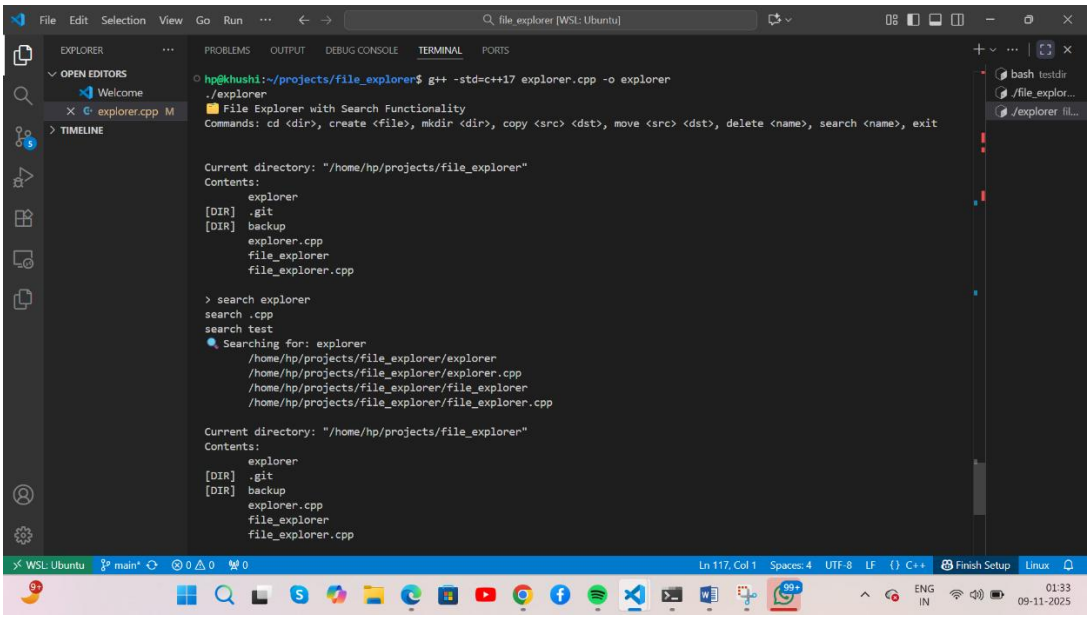
```
#include <iostream>

#include <filesystem>

#include <string>
```

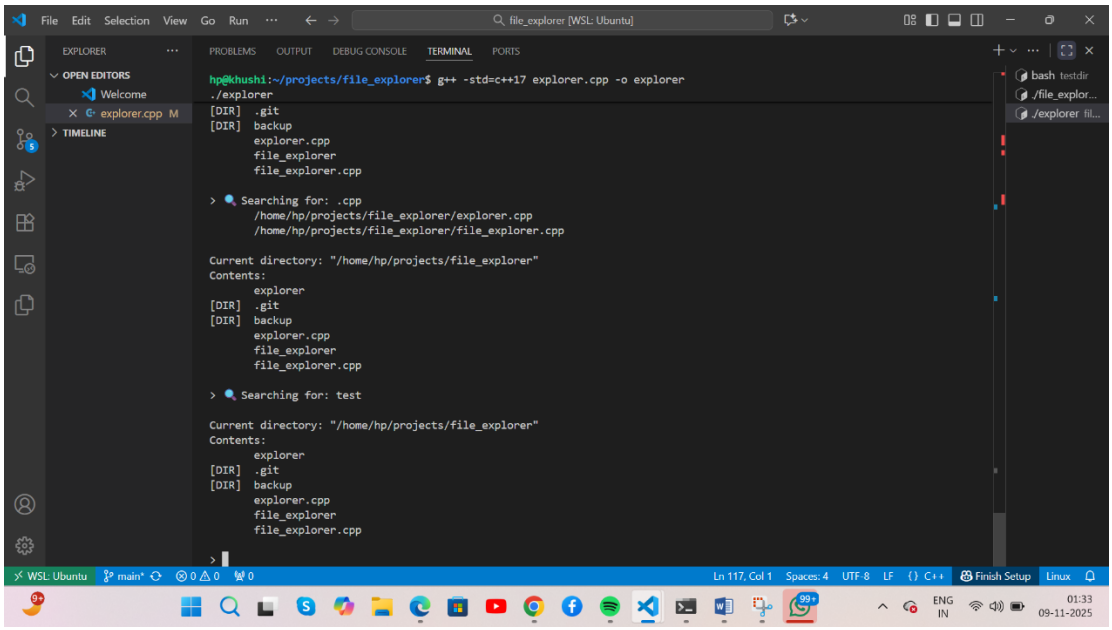
```
#include <fstream>

namespace fs = std::filesystem;

void listDirectory(const fs::path& current) {

    std::cout << "\nCurrent directory: " << current << "\n";

    std::cout << "Contents:\n";

    for (const auto& entry : fs::directory_iterator(current)) {

        std::cout << (entry.is_directory() ? "[DIR]  " : "       ");

        std::cout << entry.path().filename().string() << "\n";

    }

}

// 🔍 Recursive search function

void searchFile(const fs::path& current, const std::string& name) {

    for (const auto& entry : fs::recursive_directory_iterator(current)) {

        if (entry.path().filename().string().find(name) != std::string::npos) {

            std::cout << (entry.is_directory() ? "[DIR]  " : "       ");

            std::cout << entry.path().string() << "\n";

}}}

int main() {

    fs::path current = fs::current_path();

    std::cout << "📁 File Explorer with Search Functionality\n";

    std::cout << "Commands: cd <dir>, create <file>, mkdir <dir>, copy <src> <dst>, move <src> <dst>, delete <name>,
search <name>, exit\n\n";

    while (true) {

        listDirectory(current);

        std::cout << "\n> ";

        std::string command;

        std::cin >> command;

        if (command == "exit") break;

        else if (command == "cd") {
```

```
        std::string dir;
        std::cin >> dir;
        if (dir == ".." && current.has_parent_path()) {
            current = current.parent_path();
        } else if (fs::is_directory(current / dir)) {
            current /= dir;
        } else {
            std::cout << "❌ Directory not found.\n";
        }
    else if (command == "create") {
        std::string filename;
        std::cin >> filename;
        std::ofstream file(current / filename);
        if (file) std::cout << "✅ File created: " << filename << "\n";
        else std::cout << "❌ Failed to create file.\n";
    }
    else if (command == "mkdir") {
        std::string dirname;
        std::cin >> dirname;
        if (fs::create_directory(current / dirname))
            std::cout << "✅ Directory created: " << dirname << "\n";
        else
            std::cout << "❌ Could not create directory.\n";
    }
    else if (command == "copy") {
        std::string src, dst;
        std::cin >> src >> dst;
        try {
```

```
            fs::copy_file(current / src, current / dst, fs::copy_options::overwrite_existing);
            std::cout << "✅ File copied successfully.\n";
        } catch (...) {
            std::cout << "❌ Copy failed.\n";
}}
    else if (command == "move") {
        std::string src, dst;
        std::cin >> src >> dst;
        try {
            fs::rename(current / src, current / dst);
          std::cout << "✅ File moved/renamed.\n";
        } catch (...) {
            std::cout << "❌ Move failed.\n";
}}
    else if (command == "delete") {
        std::string target;
        std::cin >> target;
        try {
            fs::remove_all(current / target);
            std::cout << "✅ Deleted successfully.\n";
        } catch (...) {
            std::cout << "❌ Delete failed.\n"
}}
    else if (command == "search") {
        std::string name;
        std::cin >> name;
        std::cout << "🔍 Searching for: " << name << "\n";
        searchFile(current, name);
```

```
}}
    else {
        std::cout << "❌ Unknown command.\n"}}
    std::cout << "Exiting File Explorer...\n";
    return 0;
}
```





Day 5 – File Permission Management

- Added functions using `stat()` and `chmod()` to:
  - View file permissions in **rwx (read/write/execute)** format
  - Modify file permissions from the terminal

Source code:

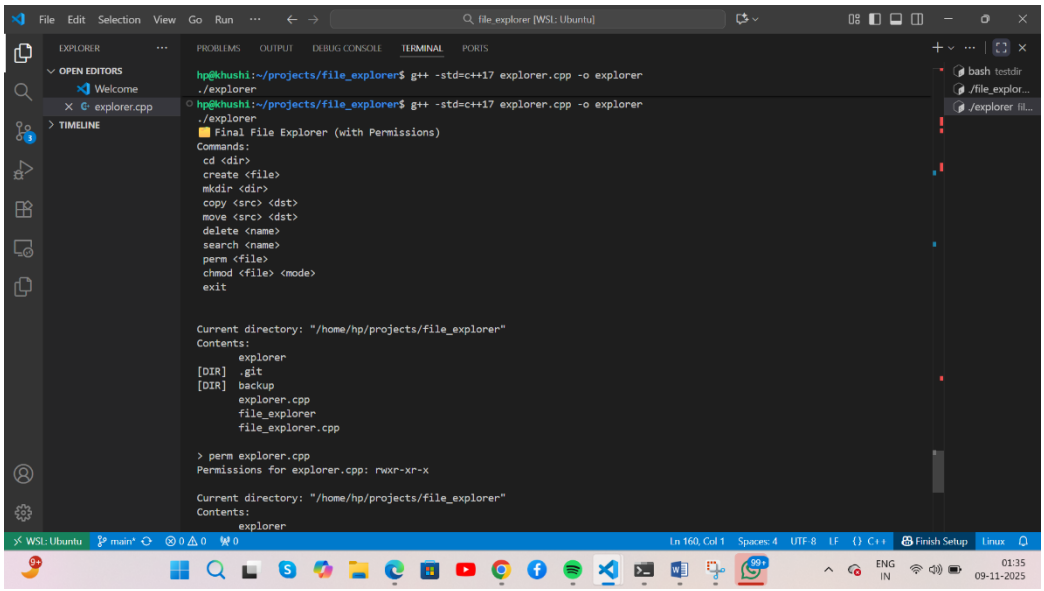
```
#include <iostream>

#include <filesystem>

#include <string>

#include <fstream>

#include <sys/stat.h>

namespace fs = std::filesystem;

void listDirectory(const fs::path& current) {

    std::cout << "\nCurrent directory: " << current << "\n";

    std::cout << "Contents:\n";

    for (const auto& entry : fs::directory_iterator(current)) {

        std::cout << (entry.is_directory() ? "[DIR]  " : "       ");

        std::cout << entry.path().filename().string() << "\n";}}

void searchFile(const fs::path& current, const std::string& name) {

    for (const auto& entry : fs::recursive_directory_iterator(current)) {

        if (entry.path().filename().string().find(name) != std::string::npos) {

            std::cout << (entry.is_directory() ? "[DIR]  " : "       ");

            std::cout << entry.path().string() << "\n";

}}

void showPermissions(const fs::path& filePath) {

    struct stat info;

    if (stat(filePath.c_str(), &info) != 0) {

        std::cout << "❌ Cannot access file: " << filePath << "\n";

        return;}

    std::cout << "Permissions for " << filePath.filename().string() << ": ";
```

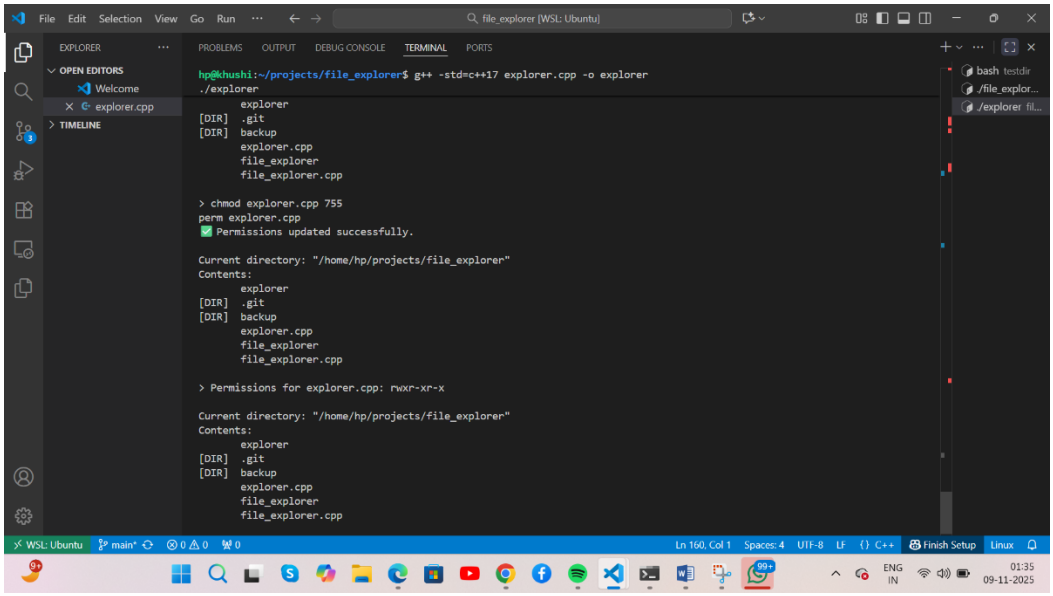
```
    std::cout << ((info.st_mode & S_IRUSR) ? "r" : "-");
    std::cout << ((info.st_mode & S_IWUSR) ? "w" : "-");
    std::cout << ((info.st_mode & S_IXUSR) ? "x" : "-");
    std::cout << ((info.st_mode & S_IRGRP) ? "r" : "-");
    std::cout << ((info.st_mode & S_IWGRP) ? "w" : "-");
    std::cout << ((info.st_mode & S_IXGRP) ? "x" : "-");
    std::cout << ((info.st_mode & S_IROTH) ? "r" : "-");
    std::cout << ((info.st_mode & S_IWOTH) ? "w" : "-");
    std::cout << ((info.st_mode & S_IXOTH) ? "x" : "-");
    std::cout << "\n";}
void changePermissions(const fs::path& filePath, int mode) {
        std::cout << "✅ Permissions updated successfully.\n";
    else
        std::cout << "❌ Failed to change permissions.\n";
}
int main() {
    fs::path current = fs::current_path();
    std::cout << "📁 Final File Explorer (with Permissions)\n";
    std::cout << "Commands:\n";
    std::cout << " cd <dir>\n create <file>\n mkdir <dir>\n copy <src> <dst>\n move <src> <dst>\n delete <name>\n search
<name>\n perm <file>\n chmod <file> <mode>\n exit\n\n";
    while (true) {
        listDirectory(current);
        std::cout << "\n> ";
        std::string command;
        std::cin >> command;
        if (command == "exit") break;
        else if (command == "cd") {
            std::string dir;
```

```
    std::cin >> dir;
    if (dir == ".." && current.has_parent_path()) {
        current = current.parent_path();
    } else if (fs::is_directory(current / dir)) {
        current /= dir;
    } else {
        std::cout << "❌ Directory not found.\n";
    }
}
else if (command == "create") {
    std::string filename;
    std::cin >> filename;
    std::ofstream file(current / filename);
    if (file) std::cout << "✅ File created: " << filename << "\n";
    else std::cout << "❌ Failed to create file.\n";
}
else if (command == "mkdir") {
    std::string dirname;
    std::cin >> dirname;
    if (fs::create_directory(current / dirname))
        std::cout << "✅ Directory created: " << dirname << "\n";
    else
        std::cout << "❌ Could not create directory.\n";
}
else if (command == "copy") {
    std::string src, dst;
    std::cin >> src >> dst;
    try {
```

```
            fs::copy_file(current / src, current / dst, fs::copy_options::overwrite_existing);
            std::cout << "✅ File copied successfully.\n";
        } catch (...) {
            std::cout << "❌ Copy failed.\n";}}
    else if (command == "move") {
        std::string src, dst;
        std::cin >> src >> dst;
        try {
            fs::rename(current / src, current / dst);
            std::cout << "✅ File moved/renamed.\n";
        } catch (...) {
            std::cout << "❌ Move failed.\n";}}
    else if (command == "delete") {
        std::string target;
        std::cin >> target;
        try {
            fs::remove_all(current / target);
            std::cout << "✅ Deleted successfully.\n";
        } catch (...) {
            std::cout << "❌ Delete failed.\n"
}}
    else if (command == "search") {
        std::string name;
        std::cin >> name;
        std::cout << "🔍 Searching for: " << name << "\n";
        searchFile(current, name);}
    else if (command == "perm") {
        std::string filename;
```

```
            std::cin >> filename;
            showPermissions(current / filename);}
        else if (command == "chmod") {
            std::string filename;
            int mode;
            std::cin >> filename >> std::oct >> mode;  // read as octal
            changePermissions(current / filename, mode);
        }


        else {
            std::cout << "❌ Unknown command.\n";
    std::cout << "Exiting File Explorer...\n";
    return 0;
}
```





5. Program Flow

1. Start Application
2. Display Current Directory
3. Accept User Commands
4. Perform Operations (List, Copy, Move, Delete, Search, etc.)
5. Display Output / Updated Directory State
6. Exit

6. Testing and Output Results

Feature	Command	Expected Output
List Files	`./explorer`	Displays all files in current directory
Create File	`create report.txt`	Creates new file
Copy File	`copy report.txt backup.txt`	File copied successfully
Move File	`move backup.txt test/`	File moved successfully
Delete File	`delete report.txt`	File deleted
Search	`search explorer`	Shows matching file paths

7. Project Repository

Project Source Code:
🔗 https://github.com/Khushi6021/FileExplorer

8. Results and Learnings

- Successfully implemented a **command-line file explorer** using Linux system calls.
- Gained practical understanding of **file handling**, **directory traversal**, and **permissions**.
- Learned to work with **GitHub** for code management and collaboration.
- Understood the connection between **C++ programming** and **operating system-level operations**.

Conclusion

This project successfully demonstrates a practical application of **C++ system programming** on a **Linux operating system**.
It provided in-depth exposure to low-level file management concepts and Linux command-line operations, strengthening the developer's understanding of operating systems and file I/O handling.