Capstone Project Report

1. Title

Project Title: File Explorer Application using C++ and Filesystem Library in Linux OS

Course: Capstone Project

Student Name: Biseswar Mohapatra

Registration No.: 2241013319

Institute: ITER, SOA University

2. Project Overview

Developing a console based file explorer application in C++ that interfaces with the Linux operating system to manage files and directories and generating each day task .

3. Project Development

Environment:

- Operating System: Ubuntu (via Docker on Windows)
- Compiler: g++ (Ubuntu 15.2.0)
- Language Standard: C++17
- Tools Used: GitHub

Day 1 – Basic File Listing

Objective: Display all files and folders in the current working directory.

Code:

```
#include <bits/stdc++.h>
#include <filesystem>
using namespace std;
namespace fs = std::filesystem;

void cmd_pwd() {
    cout << fs::current_path().string() << "\n";
}
```

```
void cmd_ls() {
  for (auto &entry : fs::directory_iterator(fs::current_path())) {
    cout << entry.path().filename().string();
    if (entry.is_directory()) cout << "/"; // mark directories
    cout << "\n";
  }
}

int main() {
  cout << "Simple File Explorer - Day 1\n";
  cout << "Commands: ls, pwd, exit\n";

  string cmd;
  while (true) {
    cout << fs::current_path().string() << " $ ";
    getline(cin, cmd);

    if (cmd == "pwd") cmd_pwd();
    else if (cmd == "ls") cmd_ls();
    else if (cmd == "exit") break;
    else if (!cmd.empty()) cout << "Unknown command\n";
  }

  cout << "Exiting File Explorer\n";
  return 0;
}
```

```
biseswar@biseswar:~/projects/file-explorer-capstone$ make
g++ -std=c++17 -O2 src/file_explorer.cpp -o file_explorer
biseswar@biseswar:~/projects/file-explorer-capstone$ ./file_explorer
Simple File Explorer - Day 1
Commands: ls, pwd, exit
```

Day 2 – Directory Navigation

Objective: Objective: Add support for ls, cd, pwd, and exit commands.

Code:

```
#include <bits/stdc++.h>
#include <filesystem>
using namespace std;
namespace fs = std::filesystem;

// ----------- Existing Commands -----------
void cmd_pwd() {
  cout << fs::current_path().string() << "\n";
}

void cmd_ls() {
```

```
    for (auto &entry : fs::directory_iterator(fs::current_path())) {
        cout << entry.path().filename().string();
        if (entry.is_directory()) cout << "/";
        cout << "\n";
    }
}
// ----------- New Day 2 Commands -----------
void cmd_cd(const string &path) {
    try {
        fs::current_path(path);
    } catch (...) {
        cout << "cd: no such directory\n";
    }
}

void cmd_mkdir(const string &dir) {
    try {
        if (fs::create_directory(dir))
            cout << "Directory created: " << dir << "\n";
        else
            cout << "mkdir: directory already exists or failed\n";
    } catch (...) {
        cout << "mkdir: failed to create directory\n";
    }
}

void cmd_touch(const string &file) {
    ofstream f(file);
    if (f)
        cout << "File created: " << file << "\n";
    else
        cout << "touch: failed to create file\n";
    f.close();
}
// ----------- Main Loop -----------
int main() {
    cout << "Simple File Explorer - Day 2\n";
    cout << "Commands: ls, pwd, cd, mkdir, touch, exit\n";

    string line;
    while (true) {
        cout << fs::current_path().string() << " $ ";
        getline(cin, line);
        stringstream ss(line);
        string cmd, arg;
        ss >> cmd >> arg;

        if (cmd == "pwd") cmd_pwd();
        else if (cmd == "ls") cmd_ls();
        else if (cmd == "cd") cmd_cd(arg);
```

```
        else if (cmd == "mkdir") cmd_mkdir(arg);
        else if (cmd == "touch") cmd_touch(arg);
        else if (cmd == "exit") break;
        else if (!cmd.empty()) cout << "Unknown command\n";
    }

    cout << "Exiting File Explorer\n";
    return 0;
}
```

```
biseswar@biseswar:~/projects/file-explorer-capstone$ ./file_explorer
Simple File Explorer - Day 2
Commands: ls, pwd, cd, mkdir, touch, exit
/home/biseswar/projects/file-explorer-capstone $ touch test.txt
File created: test.txt
/home/biseswar/projects/file-explorer-capstone $ mkdir new_folder
Directory created: new_folder
/home/biseswar/projects/file-explorer-capstone $ cd new_folder
/home/biseswar/projects/file-explorer-capstone/new_folder $ pwd
/home/biseswar/projects/file-explorer-capstone/new_folder
/home/biseswar/projects/file-explorer-capstone/new_folder $ exit
Exiting File Explorer
biseswar@biseswar:~/projects/file-explorer-capstone$
```

Day 3 – File Operations

Objective: Objective: Enable creating, deleting, copying, and moving files.

Description: Code implements touch, rm, cp, and mv commands.

Code:

```
#include <bits/stdc++.h>
#include <filesystem>
#include <chrono>
using namespace std;
namespace fs = std::filesystem;

// ---------- Existing Commands ----------
void cmd_pwd() {
    cout << fs::current_path().string() << "\n";
}

void cmd_ls() {
    for (auto &entry : fs::directory_iterator(fs::current_path())) {
        cout << entry.path().filename().string();
        if (entry.is_directory()) cout << "/";
        cout << "\n";
    }
}

void cmd_cd(const string &path) {
    try {
```

```
        fs::current_path(path);
    } catch (...) {
        cout << "cd: no such directory\n";
    }
}

void cmd_mkdir(const string &dir) {
    try {
        if (fs::create_directory(dir))
            cout << "Directory created: " << dir << "\n";
        else
            cout << "mkdir: directory already exists or failed\n";
    } catch (...) {
        cout << "mkdir: failed to create directory\n";
    }
}
void cmd_touch(const string &file) {
    ofstream f(file);
    if (f)
        cout << "File created: " << file << "\n";
    else
        cout << "touch: failed to create file\n";
    f.close();
}

// ---------- New Day 3 Commands ----------
void cmd_cp(const string &src, const string &dest) {
    try {
        fs::copy(src, dest, fs::copy_options::overwrite_existing);
        cout << "Copied " << src << " to " << dest << "\n";
    } catch (...) {
        cout << "cp: failed to copy file\n";
    }
}

void cmd_mv(const string &src, const string &dest) {
    try {
        fs::rename(src, dest);
        cout << "Moved " << src << " to " << dest << "\n";
    } catch (...) {
        cout << "mv: failed to move/rename file\n";
    }
}
void cmd_rm(const string &target) {
    try {
        if (fs::remove(target))
            cout << "Removed: " << target << "\n";
        else
            cout << "rm: no such file or directory\n";
    } catch (...) {
```

```
        cout << "rm: failed to remove\n";
    }
}

void cmd_info(const string &file) {
    try {
        if (!fs::exists(file)) {
            cout << "info: file not found\n";
            return;
        }
        auto ftime = fs::last_write_time(file);
        auto sctp = chrono::time_point_cast<chrono::system_clock::duration>(
            ftime - fs::file_time_type::clock::now() + chrono::system_clock::now()
        );
        time_t cftime = chrono::system_clock::to_time_t(sctp);
        cout << "File: " << file << "\n";
        cout << "Type: " << (fs::is_directory(file) ? "Directory" : "File") << "\n";
        cout << "Size: " << fs::file_size(file) << " bytes\n";
        cout << "Last modified: " << ctime(&cftime);
    } catch (...) {
        cout << "info: error reading file details\n";
    }
}
// ---------- Main ----------
int main() {
    cout << "Simple File Explorer - Day 3\n";
    cout << "Commands: ls, pwd, cd, mkdir, touch, cp, mv, rm, info, exit\n";

    string line;
    while (true) {
        cout << fs::current_path().string() << " $ ";
        getline(cin, line);
        stringstream ss(line);
        string cmd, arg1, arg2;
        ss >> cmd >> arg1 >> arg2;

        if (cmd == "pwd") cmd_pwd();
        else if (cmd == "ls") cmd_ls();
        else if (cmd == "cd") cmd_cd(arg1);
        else if (cmd == "mkdir") cmd_mkdir(arg1);
        else if (cmd == "touch") cmd_touch(arg1);
        else if (cmd == "cp") cmd_cp(arg1, arg2);
        else if (cmd == "mv") cmd_mv(arg1, arg2);
        else if (cmd == "rm") cmd_rm(arg1);
        else if (cmd == "info") cmd_info(arg1);
        else if (cmd == "exit") break;
        else if (!cmd.empty()) cout << "Unknown command\n";
    }

    cout << "Exiting File Explorer\n";
```

```
    return 0;
}
```

```
biseswar@biseswar:~/projects/file-explorer-capstone$ ./file_explorer
Simple File Explorer - Day 3
Commands: ls, pwd, cd, mkdir, touch, cp, mv, rm, info, exit
/home/biseswar/projects/file-explorer-capstone $ touch test.txt
File created: test.txt
/home/biseswar/projects/file-explorer-capstone $ info test.txt
File: test.txt
Type: File
Size: 0 bytes
Last modified: Sat Nov  8 10:14:19 2025
/home/biseswar/projects/file-explorer-capstone $
```

Day 4 – File Search

Objective: Objective: Search for a file recursively within the current directory.

Description: Code implements a search command using recursive_directory_iterator.

Code:

```
#include <bits/stdc++.h>
#include <filesystem>
#include <chrono>
using namespace std;
namespace fs = std::filesystem;

// ---------- Existing Commands ----------
void cmd_pwd() {
    cout << fs::current_path().string() << "\n";
}

void cmd_ls() {
    for (auto &entry : fs::directory_iterator(fs::current_path())) {
        cout << entry.path().filename().string();
        if (entry.is_directory()) cout << "/";
        cout << "\n";
    }
}

void cmd_cd(const string &path) {
    try {
        fs::current_path(path);
    } catch (...) {
        cout << "cd: no such directory\n";
    }
}
```

```
void cmd_mkdir(const string &dir) {
   try {
      if (fs::create_directory(dir))
         cout << "Directory created: " << dir << "\n";
      else
         cout << "mkdir: directory already exists or failed\n";
   } catch (...) {
      cout << "mkdir: failed to create directory\n";
   }
}

void cmd_touch(const string &file) {
   ofstream f(file);
   if (f)
      cout << "File created: " << file << "\n";
   else
      cout << "touch: failed to create file\n";
   f.close();
}

void cmd_cp(const string &src, const string &dest) {
   try {
      fs::copy(src, dest, fs::copy_options::overwrite_existing);
      cout << "Copied " << src << " to " << dest << "\n";
   } catch (...) {
      cout << "cp: failed to copy file\n";
   }
}

void cmd_mv(const string &src, const string &dest) {
   try {
      fs::rename(src, dest);
      cout << "Moved " << src << " to " << dest << "\n";
   } catch (...) {
      cout << "mv: failed to move/rename file\n";
   }
}

void cmd_rm(const string &target) {
   try {
      if (fs::remove_all(target))
         cout << "Removed: " << target << "\n";
      else
         cout << "rm: no such file or directory\n";
   } catch (...) {
      cout << "rm: failed to remove\n";
   }
}
```

```
void cmd_info(const string &file) {
    try {
        if (!fs::exists(file)) {
            cout << "info: file not found\n";
            return;
        }
        auto ftime = fs::last_write_time(file);
        auto sctp = chrono::time_point_cast<chrono::system_clock::duration>(
            ftime - fs::file_time_type::clock::now() + chrono::system_clock::now()
        );
        time_t cftime = chrono::system_clock::to_time_t(sctp);
        cout << "File: " << file << "\n";
        cout << "Type: " << (fs::is_directory(file) ? "Directory" : "File") << "\n";
        if (!fs::is_directory(file))
            cout << "Size: " << fs::file_size(file) << " bytes\n";
        cout << "Last modified: " << ctime(&cftime);
    } catch (...) {
        cout << "info: error reading file details\n";
    }
}

// ---------- New Day 4 Commands ----------
void search_recursive(const fs::path &path, const string &target) {
    try {
        for (auto &entry : fs::recursive_directory_iterator(path)) {
            if (entry.path().filename().string().find(target) != string::npos) {
                cout << entry.path().string() << "\n";
            }
        }
    } catch (...) {
        cout << "find: error during search\n";
    }
}

void cmd_find(const string &target) {
    if (target.empty()) {
        cout << "Usage: find <name>\n";
        return;
    }
    cout << "Searching for '" << target << "'...\n";
    search_recursive(fs::current_path(), target);
}

void print_tree(const fs::path &path, string prefix = "") {
    try {
        for (auto &entry : fs::directory_iterator(path)) {
            cout << prefix << "|-- " << entry.path().filename().string();
            if (entry.is_directory()) {
                cout << "/" << "\n";
                print_tree(entry.path(), prefix + "    ");
```

```
            } else {
                cout << "\n";
            }
        }
    } catch (...) {
        cout << "tree: error reading directory\n";
    }
}

void cmd_tree() {
    cout << fs::current_path().string() << "\n";
    print_tree(fs::current_path());
}

void cmd_help() {
    cout << "\nAvailable Commands:\n";
    cout << " ---------------------------------------- \n";
    cout << "pwd              - Show current directory\n";
    cout << "ls               - List files and folders\n";
    cout << "cd <dir>         - Change directory\n";
    cout << "mkdir <dir>      - Create a directory\n";
    cout << "touch <file>     - Create a file\n";
    cout << "cp <src> <dest>  - Copy file\n";
    cout << "mv <src> <dest>  - Move or rename file\n";
    cout << "rm <target>      - Delete file or folder\n";
    cout << "info <file>      - Show file info\n";
    cout << "find <name>      - Search by filename\n";
    cout << "tree             - Display folder tree\n";
    cout << "help             - Show this help message\n";
    cout << "exit             - Exit the explorer\n";
    cout << " ---------------------------------------- \n\n";
}

// ---------- Main ----------
int main() {
    cout << "Simple File Explorer - Day 4\n";
    cout << "Type 'help' for list of commands.\n";

    string line;
    while (true) {
        cout << fs::current_path().string() << " $ ";
        getline(cin, line);
        stringstream ss(line);
        string cmd, arg1, arg2;
        ss >> cmd >> arg1 >> arg2;

        if (cmd == "pwd") cmd_pwd();
        else if (cmd == "ls") cmd_ls();
        else if (cmd == "cd") cmd_cd(arg1);
        else if (cmd == "mkdir") cmd_mkdir(arg1);
```

```
        else if (cmd == "touch") cmd_touch(arg1);
        else if (cmd == "cp") cmd_cp(arg1, arg2);
        else if (cmd == "mv") cmd_mv(arg1, arg2);
        else if (cmd == "rm") cmd_rm(arg1);
        else if (cmd == "info") cmd_info(arg1);
        else if (cmd == "find") cmd_find(arg1);
        else if (cmd == "tree") cmd_tree();
        else if (cmd == "help") cmd_help();
        else if (cmd == "exit") break;
        else if (!cmd.empty()) cout << "Unknown command\n";
    }

    cout << "Exiting File Explorer\n";
    return 0;
}
```

```
biseswar@biseswar:~/projects/file-explorer-capstone$ ./file_explorer
Simple File Explorer - Day 4
Type 'help' for list of commands.
/home/biseswar/projects/file-explorer-capstone $ help

Available Commands:
----------------------------------------
pwd                 - Show current directory
ls                  - List files and folders
cd <dir>            - Change directory
mkdir <dir>         - Create a directory
touch <file>        - Create a file
cp <src> <dest>     - Copy file
mv <src> <dest>     - Move or rename file
rm <target>         - Delete file or folder
info <file>         - Show file info
find <name>         - Search by filename
tree                - Display folder tree
help                - Show this help message
exit                - Exit the explorer
----------------------------------------
```

Day 5 – File Info and Permissions

Objective: Objective: Display file size, modification date, and permissions.

Description: Code implements added history, clear ,exit confirmation.

Code:

```
#include <bits/stdc++.h>
#include <filesystem>
#include <chrono>
#include <cstdlib>
```

```
using namespace std;
namespace fs = std::filesystem;

// ---------- Command History---------- //
vector<string> command_history;

// ---------- Existing Commands---------- //
void cmd_pwd() {
   cout << fs::current_path().string() << "\n";
}

void cmd_ls() {
   for (auto &entry : fs::directory_iterator(fs::current_path())) {
      cout << entry.path().filename().string();
      if (entry.is_directory()) cout << "/";
      cout << "\n";
   }
}

void cmd_cd(const string &path) {
   try {
      fs::current_path(path);
   } catch (...) {
      cout << "cd: no such directory\n";
   }
}

void cmd_mkdir(const string &dir) {
   try {
      if (fs::create_directory(dir))
         cout << "Directory created: " << dir << "\n";
      else
         cout << "mkdir: directory already exists or failed\n";
   } catch (...) {
      cout << "mkdir: failed to create directory\n";
   }
}

void cmd_touch(const string &file) {
   ofstream f(file);
   if (f)
      cout << "File created: " << file << "\n";
   else
      cout << "touch: failed to create file\n";
   f.close();
}

void cmd_cp(const string &src, const string &dest) {
   try {
      fs::copy(src, dest, fs::copy_options::overwrite_existing);
```

```
        cout << "Copied " << src << " to " << dest << "\n";
    } catch (...) {
        cout << "cp: failed to copy file\n";
    }

}

void cmd_mv(const string &src, const string &dest) {
    try {
        fs::rename(src, dest);
        cout << "Moved " << src << " to " << dest << "\n";
    } catch (...) {
        cout << "mv: failed to move/rename file\n";
    }
}

void cmd_rm(const string &target) {
    try {
        if (fs::remove_all(target))
            cout << "Removed: " << target << "\n";
        else
            cout << "rm: no such file or directory\n";
    } catch (...) {
        cout << "rm: failed to remove\n";
    }
}

void cmd_info(const string &file) {
    try {
        if (!fs::exists(file)) {
            cout << "info: file not found\n";
            return;
        }
        auto ftime = fs::last_write_time(file);
        auto sctp = chrono::time_point_cast<chrono::system_clock::duration>(
            ftime - fs::file_time_type::clock::now() + chrono::system_clock::now()
        );
        time_t cftime = chrono::system_clock::to_time_t(sctp);
        cout << "File: " << file << "\n";
        cout << "Type: " << (fs::is_directory(file) ? "Directory" : "File") << "\n";
        if (!fs::is_directory(file))
            cout << "Size: " << fs::file_size(file) << " bytes\n";
        cout << "Last modified: " << ctime(&cftime);
    } catch (...) {
        cout << "info: error reading file details\n";
    }
}

// ---------- New Day 4 Commands ----------
void search_recursive(const fs::path &path, const string &target) {
```

```
    try {
        for (auto &entry : fs::recursive_directory_iterator(path)) {
            if (entry.path().filename().string().find(target) != string::npos) {
                cout << entry.path().string() << "\n";
            }
        }
    } catch (...) {
        cout << "find: error during search\n";
    }
}

void cmd_find(const string &target) {
    if (target.empty()) {
        cout << "Usage: find <name>\n";
        return;
    }
    cout << "Searching for '" << target << "'...\n";
    search_recursive(fs::current_path(), target);
}

void print_tree(const fs::path &path, string prefix = "") {
    try {
        for (auto &entry : fs::directory_iterator(path)) {
            cout << prefix << "|-- " << entry.path().filename().string();
            if (entry.is_directory()) {
                cout << "/" << "\n";
                print_tree(entry.path(), prefix + "    ");
            } else {
                cout << "\n";
            }
        }
    } catch (...) {
        cout << "tree: error reading directory\n";
    }
}

void cmd_tree() {
    cout << fs::current_path().string() << "\n";
    print_tree(fs::current_path());
}

void cmd_help() {
    cout << "\nAvailable Commands:\n";
    cout << " ---------------------------------------- \n";
    cout << "pwd               - Show current directory\n";
    cout << "ls                - List files and folders\n";
    cout << "cd <dir>          - Change directory\n";
    cout << "mkdir <dir>       - Create a directory\n";
    cout << "touch <file>      - Create a file\n";
    cout << "cp <src> <dest>   - Copy file\n";
```

```
    cout << "mv <src> <dest>   - Move or rename file\n";
    cout << "rm <target>       - Delete file or folder\n";
    cout << "info <file>       - Show file info\n";
    cout << "find <name>       - Search by filename\n";
    cout << "tree              - Display folder tree\n";
    cout << "clear             - Clear the screen\n";
    cout << "history           - Show all commands used\n";
    cout << "help              - Show this help message\n";
    cout << "exit          - Exit the explorer (with confirmation)\n";
    cout << " ---------------------------------------- \n\n";
}

// ---------- New Day 5 Commands ----------
void cmd_history() {
    if (command_history.empty()) {
        cout << "No commands yet.\n";
        return;
    }
    for (size_t i = 0; i < command_history.size(); ++i)
        cout << i + 1 << ": " << command_history[i] << "\n";
}

void cmd_clear() {
    system("clear");  // Clears terminal
}

// ---------- Main ----------
int main() {
    cout << "Simple File Explorer - Final Build\n";
    cout << "Type 'help' for list of commands.\n";

    string line;
    while (true) {
        cout << fs::current_path().string() << " $ ";
        getline(cin, line);
        if (line.empty()) continue;
        command_history.push_back(line);

        stringstream ss(line);
        string cmd, arg1, arg2;
        ss >> cmd >> arg1 >> arg2;

        if (cmd == "pwd") cmd_pwd();
        else if (cmd == "ls") cmd_ls();
        else if (cmd == "cd") cmd_cd(arg1);
        else if (cmd == "mkdir") cmd_mkdir(arg1);
        else if (cmd == "touch") cmd_touch(arg1);
        else if (cmd == "cp") cmd_cp(arg1, arg2);
        else if (cmd == "mv") cmd_mv(arg1, arg2);
        else if (cmd == "rm") cmd_rm(arg1);
```

```
        else if (cmd == "info") cmd_info(arg1);
        else if (cmd == "find") cmd_find(arg1);
        else if (cmd == "tree") cmd_tree();
        else if (cmd == "history") cmd_history();
        else if (cmd == "clear") cmd_clear();
        else if (cmd == "help") cmd_help();
        else if (cmd == "exit") {
            cout << "Are you sure you want to exit? (y/n): ";
            string confirm;
            getline(cin, confirm);
            if (confirm == "y" || confirm == "Y") break;
        }
        else if (!cmd.empty()) cout << "Unknown command\n";
    }

    cout << "Exiting File Explorer. Goodbye!\n";
    return 0;
}
```

```
biseswar@biseswar:~/projects/file-explorer-capstone$ ./file_explorer
Simple File Explorer - Final Build
Type 'help' for list of commands.
/home/biseswar/projects/file-explorer-capstone $ help

Available Commands:
----------------------------------------
pwd                 - Show current directory
ls                  - List files and folders
cd <dir>            - Change directory
mkdir <dir>         - Create a directory
touch <file>        - Create a file
cp <src> <dest>     - Copy file
mv <src> <dest>     - Move or rename file
rm <target>         - Delete file or folder
info <file>         - Show file info
find <name>         - Search by filename
tree                - Display folder tree
clear               - Clear the screen
history             - Show all commands used
help                - Show this help message
exit                - Exit the explorer (with confirmation)
----------------------------------------

/home/biseswar/projects/file-explorer-capstone $ exit
Are you sure you want to exit? (y/n): y
Exiting File Explorer. Goodbye!
```

5. Conclusion

The File Explorer Application successfully demonstrates file and directory manipulation, search, info, and permission management using C++17's filesystem library. The modular design allows future expansion into GUI-based explorers.