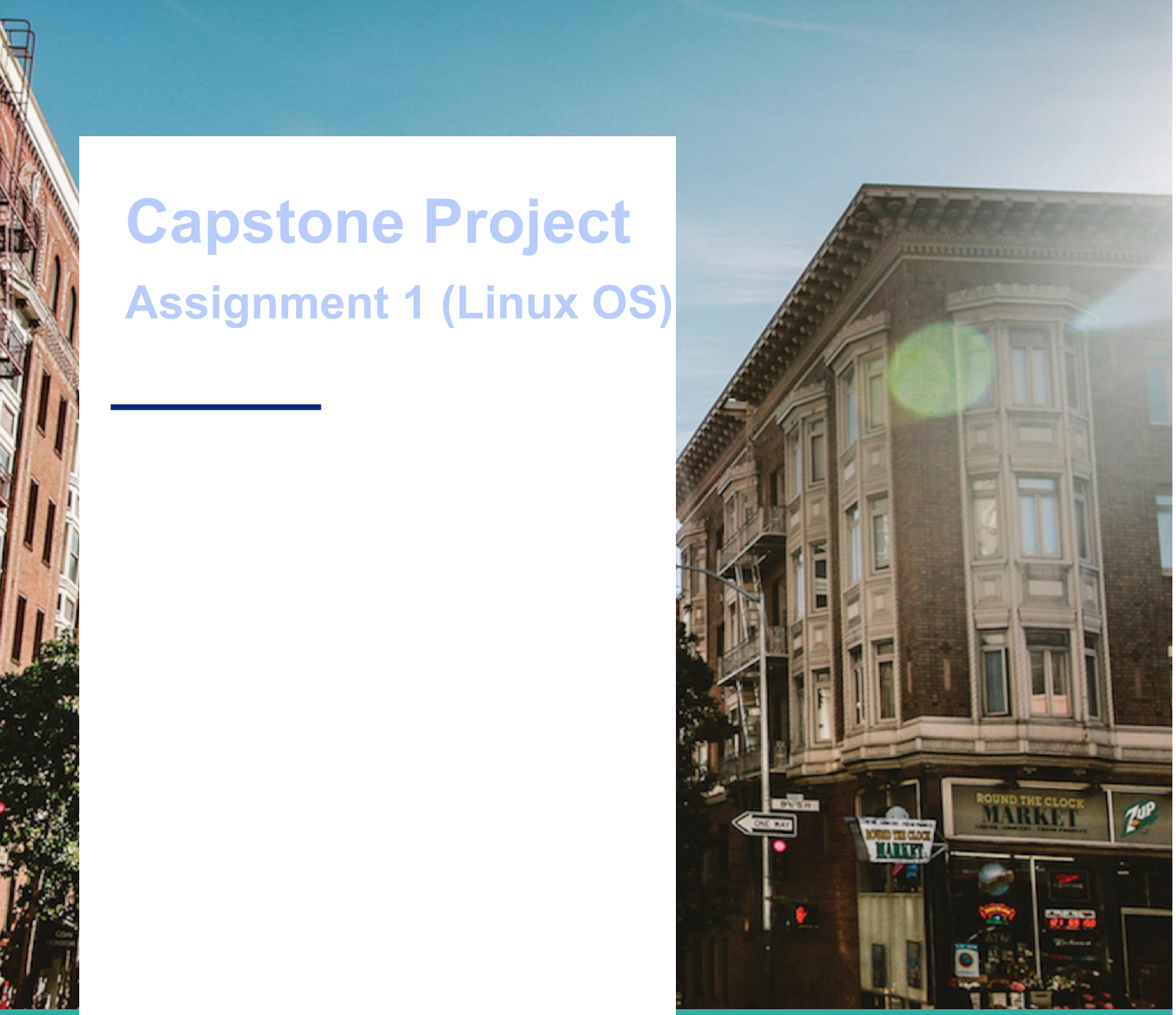

Capstone Project

Assignment 1 (Linux OS)

9 NOVEMBER 2025

Institute of Technical Education and Research
Authored by: Ashish Singh Chauhan
Wipro batch 5
Regd no : 2241003022

Assignment 1(Linux OS)

File Explorer Application

Objective: Develop a console-based file explorer application in C++ that interfaces with the Linux operating system to manage files and directories.

Summary: In this assignment, we built a command-line file explorer for Linux environments using C++17, compiled and executed through WSL on Windows 11. The project mimics basic file-system functionality similar to standard Linux commands.

CODE :

```
#include <bits/stdc++.h>
#include <filesystem>
#include <iomanip>
#include <pwd.h>
#include <grp.h>
#include <sys/stat.h>
#include <unistd.h>

namespace fs = std::filesystem;

// ---------- Helpers ----------
std::string perms_to_string(fs::perms p) {
    auto bit = [&](fs::perms b, char c){ return ( (p & b) != fs::perms::none ) ? c : '-'; };
    std::string s;
    s += bit(fs::perms::owner_read,  'r');
    s += bit(fs::perms::owner_write, 'w');
    s += bit(fs::perms::owner_exec,  'x');
    s += bit(fs::perms::group_read,  'r');
```

```
    s += bit(fs::perms::group_write, 'w');
    s += bit(fs::perms::group_exec,  'x');
    s += bit(fs::perms::others_read, 'r');
    s += bit(fs::perms::others_write,'w');
    s += bit(fs::perms::others_exec, 'x');
    return s;
}

std::string human_size(uintmax_t bytes) {
    const char* suffix[] = {"B","KB","MB","GB","TB"};
    double c = bytes; int i = 0;
    while (c >= 1024 && i < 4) { c /= 1024; ++i; }
    std::ostringstream os; os<<std::fixed<<std::setprecision((i==0)?0:1)<<c<<suffix[i];
    return os.str();
}

std::string owner_name(const fs::path& p) {
    struct stat st{};
    if (lstat(p.c_str(), &st) == -1) return "?";
    passwd* pw = getpwuid(st.st_uid);
    group*  gr = getgrgid(st.st_gid);
    std::ostringstream os;
    os << (pw?pw->pw_name:std::to_string(st.st_uid)) << ":" << (gr?gr-
>gr_name:std::to_string(st.st_gid));
    return os.str();
}

void print_entry(const fs::directory_entry& e) {
    auto st = fs::symlink_status(e.path());
    auto type = fs::is_directory(st) ? 'd' : fs::is_symlink(st) ? 'l' : '-';
    auto perm = perms_to_string(st.permissions());
    uintmax_t size = 0;
    if (fs::is_regular_file(st)) {
```

```
        std::error_code ec; size = fs::file_size(e.path(), ec);
    }

    // Convert filesystem clock -> system_clock to get time_t (portable)
    auto ftime = fs::last_write_time(e.path());
    auto sctp = std::chrono::time_point_cast<std::chrono::system_clock::duration>(
        ftime - fs::file_time_type::clock::now() + std::chrono::system_clock::now()
    );
    std::time_t tt = std::chrono::system_clock::to_time_t(sctp);
    std::tm tm = *std::localtime(&tt);

    std::cout << type << perm << " "
              << std::setw(10) << owner_name(e.path()) << " "
              << std::setw(8) << human_size(size) << "  "
              << std::put_time(&tm, "%Y-%m-%d %H:%M") << "  "
              << e.path().filename().string() << "\n";
}

bool iequals(const std::string& a, const std::string& b) {
    if (a.size()!=b.size()) return false;
    for (size_t i=0;i<a.size();++i)
        if (std::tolower(a[i]) != std::tolower(b[i])) return false;
    return true;
}

bool contains_icase(const std::string& hay, const std::string& needle) {
    auto H = hay; auto N = needle;
    std::transform(H.begin(),H.end(),H.begin(),::tolower);
    std::transform(N.begin(),N.end(),N.begin(),::tolower);
    return H.find(N) != std::string::npos;
}

void help() {
```

```
    std::cout <<
R"(Commands:
  ls [path]              list directory
  cd <path>              change directory
  pwd                    print working directory
  tree [depth]           tree view (default depth=2)
  mkdir <name>           create directory
  touch <name>           create empty file (or update mtime)
  rm <path>              remove file
  rmdir <path>           remove directory recursively
  cp <src> <dst>         copy (file or directory)
  mv <src> <dst>         move/rename
  open <file>            print file (first 200 lines)
  search <name-frag>     search recursively by name
  chmod <octal> <path>   set permissions (e.g., 755)
  perms [path]           show permissions entries
  help                   show this help
  exit                   quit
)";
}

void list_dir(const fs::path& p) {
    std::vector<fs::directory_entry> items;
    std::error_code ec;
    for (auto& e : fs::directory_iterator(p, ec)) items.push_back(e);
    if (ec) { std::cerr << "ls: " << ec.message() << "\n"; return; }
    std::sort(items.begin(), items.end(),
              [](const auto& a, const auto& b){
                if (a.is_directory() != b.is_directory())
                    return a.is_directory() && !b.is_directory();
                return a.path().filename().string() < b.path().filename().string();
              });
    for (auto& e : items) print_entry(e);
```

```
}

void tree(const fs::path& root, int max_depth, int depth=0) {
   if (depth > max_depth) return;
   std::error_code ec;
   for (auto& e : fs::directory_iterator(root, ec)) {
      if (ec) { std::cerr << "tree: " << ec.message() << "\n"; return; }
      for (int i=0;i<depth;i++) std::cout << "  ";
      std::cout << "|- " << e.path().filename().string() << "\n";
      if (e.is_directory()) tree(e.path(), max_depth, depth+1);
   }
}

void touch(const fs::path& p) {
   std::error_code ec;
   if (!fs::exists(p)) {
      std::ofstream f(p);
      if (!f) { std::cerr<<"touch: cannot create\n"; return; }
   } else {
      auto now = fs::file_time_type::clock::now();   // fixed
      fs::last_write_time(p, now, ec);
      if (ec) { std::cerr<<"touch: "<<ec.message()<<"\n"; }
   }
}

void copy_any(const fs::path& src, const fs::path& dst) {
   std::error_code ec;
   if (fs::is_directory(src)) {
      fs::create_directories(dst, ec);
      fs::copy(src, dst, fs::copy_options::recursive | fs::copy_options::overwrite_existing,
ec);
   } else {
      fs::copy_file(src, dst, fs::copy_options::overwrite_existing, ec);
```

```
    }
    if (ec) std::cerr << "cp: " << ec.message() << "\n";
}

void show_file(const fs::path& p) {
    std::ifstream in(p);
    if (!in) { std::cerr << "open: cannot open file\n"; return; }
    std::string line; int n=0;
    while (n<200 && std::getline(in,line)) { std::cout << line << "\n"; ++n; }
    if (!in.eof()) std::cout << "...(truncated)\n";
}

void search_name(const fs::path& root, const std::string& pat) {
    std::error_code ec;
    for (auto it = fs::recursive_directory_iterator(root, ec);
         it != fs::recursive_directory_iterator(); ++it) {
        if (ec) { std::cerr << "search: " << ec.message() << "\n"; break; }
        if (contains_icase(it->path().filename().string(), pat)) {
            std::cout << it->path().string() << "\n";
        }
    }
}

void show_perms(const fs::path& p) {
    if (!fs::exists(p)) { std::cerr<<"perms: path not found\n"; return; }
    if (fs::is_directory(p)) {
        for (auto& e : fs::directory_iterator(p)) print_entry(e);
    } else {
        print_entry(fs::directory_entry(p));
    }
}

void chmod_octal(const fs::path& p, const std::string& oct) {
```

```
    if (oct.size()<3 || oct.size()>4 || !std::all_of(oct.begin(),oct.end(),::isdigit)) {
        std::cerr << "chmod: use octal like 755 or 0644\n"; return;
    }
    unsigned mode = std::stoul(oct, nullptr, 8);
    fs::perms perm = static_cast<fs::perms>(mode);
    std::error_code ec;
    fs::permissions(p, perm, ec);
    if (ec) std::cerr << "chmod: " << ec.message() << "\n";
}

// ---------- REPL ----------
int main() {
    std::cout << "File Explorer (C++17, Linux/WSL)\n";
    help();
    fs::path cwd = fs::current_path();

    std::string line;
    while (true) {
        std::cout << "\n[" << cwd.string() << "]$ ";
        if (!std::getline(std::cin, line)) break;
        std::istringstream iss(line);
        std::string cmd; iss >> cmd;
        if (cmd.empty()) continue;

        try {
            if (cmd=="ls") {
                std::string p; iss >> p;
                list_dir(p.empty()?cwd:fs::path(p).is_absolute()?fs::path(p):(cwd/p));
            } else if (cmd=="pwd") {
                std::cout << cwd.string() << "\n";
            } else if (cmd=="cd") {
                std::string p; iss >> p;
                if (p.empty()) { std::cerr<<"cd: path required\n"; continue; }
```

```
            fs::path np = fs::path(p).is_absolute()?fs::path(p):(cwd/p);
            if (fs::exists(np) && fs::is_directory(np)) { cwd = fs::canonical(np);
fs::current_path(cwd); }
            else std::cerr<<"cd: not a directory\n";
        } else if (cmd=="tree") {
            int d=2; iss>>d; if (d<0) d=0; tree(cwd,d);
        } else if (cmd=="mkdir") {
            std::string n; iss>>n; if(n.empty()){std::cerr<<"mkdir: name
required\n";continue;}
            std::error_code ec; fs::create_directories(cwd/n, ec); if(ec) std::cerr<<"mkdir:
"<<ec.message()<<"\n";
        } else if (cmd=="touch") {
            std::string n; iss>>n; if(n.empty()){std::cerr<<"touch: name
required\n";continue;}
            touch(cwd/n);
        } else if (cmd=="rm") {
            std::string p; iss>>p; if(p.empty()){std::cerr<<"rm: path required\n";continue;}
            std::error_code ec; fs::remove(cwd/p, ec); if(ec) std::cerr<<"rm:
"<<ec.message()<<"\n";
        } else if (cmd=="rmdir") {
            std::string p; iss>>p; if(p.empty()){std::cerr<<"rmdir: path
required\n";continue;}
            std::error_code ec; fs::remove_all(cwd/p, ec); if(ec) std::cerr<<"rmdir:
"<<ec.message()<<"\n";
        } else if (cmd=="cp") {
            std::string s,d; iss>>s>>d; if(d.empty()){std::cerr<<"cp: src dst
required\n";continue;}
            copy_any(fs::path(s).is_absolute()?fs::path(s):(cwd/s),
                fs::path(d).is_absolute()?fs::path(d):(cwd/d));
        } else if (cmd=="mv") {
            std::string s,d; iss>>s>>d; if(d.empty()){std::cerr<<"mv: src dst
required\n";continue;}
            std::error_code ec;
```

```
            fs::rename(fs::path(s).is_absolute()?fs::path(s):(cwd/s),
                    fs::path(d).is_absolute()?fs::path(d):(cwd/d), ec);
            if(ec) std::cerr<<"mv: "<<ec.message()<<"\n";
        } else if (cmd=="open") {
            std::string f; iss>>f; if(f.empty()){std::cerr<<"open: file required\n";continue;}
            show_file(fs::path(f).is_absolute()?fs::path(f):(cwd/f));
        } else if (cmd=="search") {
            std::string pat; iss>>pat; if(pat.empty()){std::cerr<<"search: name
required\n";continue;}
            search_name(cwd, pat);
        } else if (cmd=="chmod") {
            std::string oct, p; iss>>oct>>p;
            if(p.empty()){std::cerr<<"chmod: <octal> <path>\n";continue;}
            chmod_octal(fs::path(p).is_absolute()?fs::path(p):(cwd/p), oct);
        } else if (cmd=="perms") {
            std::string p; iss>>p; fs::path t =
p.empty()?cwd:(fs::path(p).is_absolute()?fs::path(p):(cwd/p));
            show_perms(t);
        } else if (cmd=="help") {
            help();
        } else if (cmd=="exit" || cmd=="quit") {
            break;
        } else {
            std::cerr << "Unknown command. Type 'help'.\n";
        }
      } catch(const std::exception& ex) {
        std::cerr << "error: " << ex.what() << "\n";
      }
   }
   return 0;
}
```

Screenshot of code:

```cpp
#include <bits/stdc++.h>
#include <filesystem>
#include <iomanip>
#include <pwd.h>
#include <grp.h>
#include <sys/stat.h>
#include <unistd.h>

namespace fs = std::filesystem;

// ---------- Helpers ----------
std::string perms_to_string(fs::perms p) {
    auto bit = [&](fs::perms b, char c){ return ( (p & b) != fs::perms::none ) ? c : '-'; };
    std::string s;
    s += bit(fs::perms::owner_read,  'r');
    s += bit(fs::perms::owner_write, 'w');
    s += bit(fs::perms::owner_exec,  'x');
    s += bit(fs::perms::group_read,  'r');
    s += bit(fs::perms::group_write, 'w');
    s += bit(fs::perms::group_exec,  'x');
    s += bit(fs::perms::others_read, 'r');
    s += bit(fs::perms::others_write,'w');
    s += bit(fs::perms::others_exec, 'x');
    return s;
}

std::string human_size(uintmax_t bytes) {
    const char* suffix[] = {"B","KB","MB","GB","TB"};
    double c = bytes; int i = 0;
    while (c >= 1024 && i < 4) { c /= 1024; ++i; }
    std::ostringstream os; os<<std::fixed<<std::setprecision((i==0)?0:1)<<c<<suffix[i];
    return os.str();
}

std::string owner_name(const fs::path& p) {
    struct stat st{};
    if (lstat(p.c_str(), &st) == -1) return "?";
```

```
38      passwd* pw = getpwuid(st.st_uid);
39      group*  gr = getgrgid(st.st_gid);
40      std::ostringstream os;
41      os << (pw?pw->pw_name:std::to_string(st.st_uid)) << ":" << (gr?gr->gr_name:std::to_string(st.st_gid));
42      return os.str();
43  }
44
45  void print_entry(const fs::directory_entry& e) {
46      auto st = fs::symlink_status(e.path());
47      auto type = fs::is_directory(st) ? 'd' : fs::is_symlink(st) ? 'l' : '-';
48      auto perm = perms_to_string(st.permissions());
49      uintmax_t size = 0;
50      if (fs::is_regular_file(st)) {
51          std::error_code ec; size = fs::file_size(e.path(), ec);
52      }
53
54      // Convert filesystem clock -> system_clock to get time_t (portable)
55      auto ftime = fs::last_write_time(e.path());
56      auto sctp = std::chrono::time_point_cast<std::chrono::system_clock::duration>(
57          ftime - fs::file_time_type::clock::now() + std::chrono::system_clock::now()
58      );
59      std::time_t tt = std::chrono::system_clock::to_time_t(sctp);
60      std::tm tm = *std::localtime(&tt);
61
62      std::cout << type << perm << " "
63                << std::setw(10) << owner_name(e.path()) << " "
64                << std::setw(8) << human_size(size) << "  "
65                << std::put_time(&tm, "%Y-%m-%d %H:%M") << "  "
66                << e.path().filename().string() << "\n";
67  }
68
69  bool iequals(const std::string& a, const std::string& b) {
70      if (a.size()!=b.size()) return false;
71      for (size_t i=0;i<a.size();++i)
72          if (std::tolower(a[i]) != std::tolower(b[i])) return false;
73      return true;
```

```
74  }
75
76  bool contains_icase(const std::string& hay, const std::string& needle) {
77      auto H = hay; auto N = needle;
78      std::transform(H.begin(),H.end(),H.begin(),::tolower);
79      std::transform(N.begin(),N.end(),N.begin(),::tolower);
80      return H.find(N) != std::string::npos;
81  }
82
83  void help() {
84      std::cout <<
85  R"(Commands:
86    ls [path]                 list directory
87    cd <path>                 change directory
88    pwd                       print working directory
89    tree [depth]              tree view (default depth=2)
90    mkdir <name>              create directory
91    touch <name>              create empty file (or update mtime)
92    rm <path>                 remove file
93    rmdir <path>              remove directory recursively
94    cp <src> <dst>            copy (file or directory)
95    mv <src> <dst>            move/rename
96    open <file>               print file (first 200 lines)
97    search <name-frag>        search recursively by name
98    chmod <octal> <path>      set permissions (e.g., 755)
99    perms [path]              show permissions entries
100   help                      show this help
101   exit                      quit
102 )";
103 }
104
105 void list_dir(const fs::path& p) {
106     std::vector<fs::directory_entry> items;
107     std::error_code ec;
108     for (auto& e : fs::directory_iterator(p, ec)) items.push_back(e);
109     if (ec) { std::cerr << "ls: " << ec.message() << "\n"; return; }
```

```
110     std::sort(items.begin(), items.end(),
111               [](const auto& a, const auto& b){
112                   if (a.is_directory() != b.is_directory())
113                       return a.is_directory() && !b.is_directory();
114                   return a.path().filename().string() < b.path().filename().string();
115               });
116     for (auto& e : items) print_entry(e);
117 }
118 
119 void tree(const fs::path& root, int max_depth, int depth=0) {
120     if (depth > max_depth) return;
121     std::error_code ec;
122     for (auto& e : fs::directory_iterator(root, ec)) {
123         if (ec) { std::cerr << "tree: " << ec.message() << "\n"; return; }
124         for (int i=0;i<depth;i++) std::cout << "  ";
125         std::cout << "|- " << e.path().filename().string() << "\n";
126         if (e.is_directory()) tree(e.path(), max_depth, depth+1);
127     }
128 }
129 
130 void touch(const fs::path& p) {
131     std::error_code ec;
132     if (!fs::exists(p)) {
133         std::ofstream f(p);
134         if (!f) { std::cerr<<"touch: cannot create\n"; return; }
135     } else {
136         auto now = fs::file_time_type::clock::now();   // fixed
137         fs::last_write_time(p, now, ec);
138         if (ec) { std::cerr<<"touch: "<<ec.message()<<"\n"; }
139     }
140 }
141 
142 void copy_any(const fs::path& src, const fs::path& dst) {
143     std::error_code ec;
144     if (fs::is_directory(src)) {
145         fs::create_directories(dst, ec);
```

```
146         fs::copy(src, dst, fs::copy_options::recursive | fs::copy_options::overwrite_existing, ec);
147     } else {
148         fs::copy_file(src, dst, fs::copy_options::overwrite_existing, ec);
149     }
150     if (ec) std::cerr << "cp: " << ec.message() << "\n";
151 }
152 
153 void show_file(const fs::path& p) {
154     std::ifstream in(p);
155     if (!in) { std::cerr << "open: cannot open file\n"; return; }
156     std::string line; int n=0;
157     while (n<200 && std::getline(in,line)) { std::cout << line << "\n"; ++n; }
158     if (!in.eof()) std::cout << "...(truncated)\n";
159 }
160 
161 void search_name(const fs::path& root, const std::string& pat) {
162     std::error_code ec;
163     for (auto it = fs::recursive_directory_iterator(root, ec);
164          it != fs::recursive_directory_iterator(); ++it) {
165         if (ec) { std::cerr << "search: " << ec.message() << "\n"; break; }
166         if (contains_icase(it->path().filename().string(), pat)) {
167             std::cout << it->path().string() << "\n";
168         }
169     }
170 }
171 
172 void show_perms(const fs::path& p) {
173     if (!fs::exists(p)) { std::cerr<<"perms: path not found\n"; return; }
174     if (fs::is_directory(p)) {
175         for (auto& e : fs::directory_iterator(p)) print_entry(e);
176     } else {
177         print_entry(fs::directory_entry(p));
178     }
179 }
```

```cpp
void chmod_octal(const fs::path& p, const std::string& oct) {
    if (oct.size()<3 || oct.size()>4 || !std::all_of(oct.begin(),oct.end(),::isdigit)) {
        std::cerr << "chmod: use octal like 755 or 0644\n"; return;
    }
    unsigned mode = std::stoul(oct, nullptr, 8);
    fs::perms perm = static_cast<fs::perms>(mode);
    std::error_code ec;
    fs::permissions(p, perm, ec);
    if (ec) std::cerr << "chmod: " << ec.message() << "\n";
}

// ---------- REPL ----------
int main() {
    std::cout << "File Explorer (C++17, Linux/WSL)\n";
    help();
    fs::path cwd = fs::current_path();

    std::string line;
    while (true) {
        std::cout << "\n[" << cwd.string() << "]$ ";
        if (!std::getline(std::cin, line)) break;
        std::istringstream iss(line);
        std::string cmd; iss >> cmd;
        if (cmd.empty()) continue;

        try {
            if (cmd=="ls") {
                std::string p; iss >> p;
                list_dir(p.empty()?cwd:fs::path(p).is_absolute()?fs::path(p):(cwd/p));
            } else if (cmd=="pwd") {
                std::cout << cwd.string() << "\n";
            } else if (cmd=="cd") {
                std::string p; iss >> p;
                if (p.empty()) { std::cerr<<"cd: path required\n"; continue; }
                fs::path np = fs::path(p).is_absolute()?fs::path(p):(cwd/p);
                if (fs::exists(np) && fs::is_directory(np)) { cwd = fs::canonical(np); fs::current_path(cwd); }
                else std::cerr<<"cd: not a directory\n";
```

```
            } else if (cmd=="tree") {
                int d=2; iss>>d; if (d<0) d=0; tree(cwd,d);
            } else if (cmd=="mkdir") {
                std::string n; iss>>n; if(n.empty()){std::cerr<<"mkdir: name required\n";continue;}
                std::error_code ec; fs::create_directories(cwd/n, ec); if(ec) std::cerr<<"mkdir: "<<ec.message()<<"\n";
            } else if (cmd=="touch") {
                std::string n; iss>>n; if(n.empty()){std::cerr<<"touch: name required\n";continue;}
                touch(cwd/n);
            } else if (cmd=="rm") {
                std::string p; iss>>p; if(p.empty()){std::cerr<<"rm: path required\n";continue;}
                std::error_code ec; fs::remove(cwd/p, ec); if(ec) std::cerr<<"rm: "<<ec.message()<<"\n";
            } else if (cmd=="rmdir") {
                std::string p; iss>>p; if(p.empty()){std::cerr<<"rmdir: path required\n";continue;}
                std::error_code ec; fs::remove_all(cwd/p, ec); if(ec) std::cerr<<"rmdir: "<<ec.message()<<"\n";
            } else if (cmd=="cp") {
                std::string s,d; iss>>s>>d; if(d.empty()){std::cerr<<"cp: src dst required\n";continue;}
                copy_any(fs::path(s).is_absolute()?fs::path(s):(cwd/s),
                         fs::path(d).is_absolute()?fs::path(d):(cwd/d));
            } else if (cmd=="mv") {
                std::string s,d; iss>>s>>d; if(d.empty()){std::cerr<<"mv: src dst required\n";continue;}
                std::error_code ec;
                fs::rename(fs::path(s).is_absolute()?fs::path(s):(cwd/s),
                           fs::path(d).is_absolute()?fs::path(d):(cwd/d), ec);
                if(ec) std::cerr<<"mv: "<<ec.message()<<"\n";
            } else if (cmd=="open") {
                std::string f; iss>>f; if(f.empty()){std::cerr<<"open: file required\n";continue;}
                show_file(fs::path(f).is_absolute()?fs::path(f):(cwd/f));
            } else if (cmd=="search") {
                std::string pat; iss>>pat; if(pat.empty()){std::cerr<<"search: name required\n";continue;}
                search_name(cwd, pat);
            } else if (cmd=="chmod") {
                std::string oct, p; iss>>oct>>p;
                if(p.empty()){std::cerr<<"chmod: <octal> <path>\n";continue;}
                chmod_octal(fs::path(p).is_absolute()?fs::path(p):(cwd/p), oct);
            } else if (cmd=="perms") {
                std::string p; iss>>p; fs::path t = p.empty()?cwd:(fs::path(p).is_absolute()?fs::path(p):(cwd/p));
                show_perms(t);
            } else if (cmd=="help") {
                help();
            } else if (cmd=="exit" || cmd=="quit") {
                break;
            } else {
                std::cerr << "Unknown command. Type 'help'.\n";
            }
        } catch(const std::exception& ex) {
            std::cerr << "error: " << ex.what() << "\n";
        }
    }
    return 0;
}
```

Screenshot of outputs:

```
[/home/chauh/projects/file-explorer]$ pwd
/home/chauh/projects/file-explorer
```

pwd — show current working directory

```
[/home/chauh/projects/file-explorer]$ help
Commands:
  ls [path]                 list directory
  cd <path>                 change directory
  pwd                       print working directory
  tree [depth]              tree view (default depth=2)
  mkdir <name>              create directory
  touch <name>              create empty file (or update mtime)
  rm <path>                 remove file
  rmdir <path>              remove directory recursively
  cp <src> <dst>            copy (file or directory)
  mv <src> <dst>            move/rename
  open <file>               print file (first 200 lines)
  search <name-frag>        search recursively by name
  chmod <octal> <path>      set permissions (e.g., 755)
  perms [path]              show permissions entries
  help                      show this help
  exit                      quit
```

help — show commands

```
[/home/chauh/projects/file-explorer]$ ls
drwxr-xr-x chauh:chauh       0B  2025-11-08 17:44  .git
drwxr-xr-x chauh:chauh       0B  2025-11-08 18:48  src
-rw-r--r-- chauh:chauh     309B  2025-11-08 18:06  Makefile
-rwxr-xr-x chauh:chauh   94.7KB  2025-11-08 18:48  file_explorer
```

ls — list files

```
[/home/chauh/projects/file-explorer]$ cd src

[/home/chauh/projects/file-explorer/src]$ cd ..
```

cd <path> — change directory

```
[/home/chauh/projects/file-explorer]$ tree
|- file_explorer
|- .git
  |- description
  |- hooks
    |- update.sample
    |- sendemail-validate.sample
    |- commit-msg.sample
    |- pre-receive.sample
    |- pre-applypatch.sample
    |- push-to-checkout.sample
    |- applypatch-msg.sample
    |- pre-merge-commit.sample
    |- pre-push.sample
    |- prepare-commit-msg.sample
    |- pre-rebase.sample
    |- fsmonitor-watchman.sample
    |- pre-commit.sample
    |- post-update.sample
  |- refs
    |- heads
    |- tags
  |- branches
  |- HEAD
  |- objects
    |- info
    |- pack
  |- config
  |- info
    |- exclude
|- Makefile
|- src
  |- main.o
  |- main.cpp
```

tree

```
[/home/chauh/projects/file-explorer]$ tree 1
|- file_explorer
|- .git
  |- description
  |- hooks
  |- refs
  |- branches
  |- HEAD
  |- objects
  |- config
  |- info
|- Makefile
|- src
  |- main.o
  |- main.cpp
```

Tree[depth]

```
[/home/chauh/projects/file-explorer]$ mkdir ashish

[/home/chauh/projects/file-explorer]$ cd ashish
```

mkdir <name> - make directory

```
[/home/chauh/projects/file-explorer/ashish]$ touch ashish.txt
```

touch <name> — create/update file timestamp

```
[/home/chauh/projects/file-explorer/ashish]$ open ashish.txt
Hi , im Ashish Singh Chauhan from iter college in Wipro batch 5 and this is testing of project , hope it works
 ciao !!!
```

open <file> — show first 200 lines

```
[/home/chauh/projects/file-explorer/ashish]$ cp ashish.txt backup.txt
```

cp <src> <dst> — copy

```
[/home/chauh/projects/file-explorer/ashish]$ open backup.txt
Hi , im Ashish Singh Chauhan from iter college in Wipro batch 5 and this is testing of project , hope it works
 ciao !!!
```

```
[/home/chauh/projects/file-explorer/ashish]$ mv backup.txt copy.txt
```

mv <src> <dst> — move/rename

```
[/home/chauh/projects/file-explorer/ashish]$ rm copy.txt

[/home/chauh/projects/file-explorer/ashish]$ rm ashish.txt
```

rm <file> — delete file

```
[/home/chauh/projects/file-explorer]$ rmdir ashish
```

rmdir <dir> — delete directory (recursive)

```
[/home/chauh/projects/file-explorer]$ search main
/home/chauh/projects/file-explorer/src/main.o
/home/chauh/projects/file-explorer/src/main.cpp
```

search <name-frag> — recursive search

```
[/home/chauh/projects/file-explorer]$ perms
-rwxr-xr-x chauh:chauh   94.7KB  2025-11-08 18:48  file_explorer
drwxr-xr-x chauh:chauh       0B  2025-11-08 17:44  .git
-rw-r--r-- chauh:chauh     309B  2025-11-08 18:06  Makefile
drwxr-xr-x chauh:chauh       0B  2025-11-08 18:48  src
```

perms [path] — show file perms

```
[/home/chauh/projects/file-explorer]$ perms src/main.cpp
-rw-r--r-- chauh:chauh   10.7KB  2025-11-08 18:21  main.cpp
```

perms [path] — show file perms of specific file