Checkout and merge functionalities

- `checkout` as a tool for navigating through the project's timeline and
- `merge` as the tool for weaving different timelines together.

The code:

```
void checkout(const std::string& target) {
    std::string commit_hash;
    if (fs::exists(refs_dir + "/" + target)) {
        commit_hash = read_file(refs_dir + "/" + target);
        write_file(head_file, "ref: refs/" + target);
    } else {
        commit_hash = target;
        if (!fs::exists(objects_dir + "/" + commit_hash)) {
            std::cout << "Error: Commit not found\n";
            return;
        }
    }
    // Clear working directory
    for (auto& [file, _] : staging_area) {
        if (fs::exists(file)) fs::remove(file);
    }
    staging_area.clear();
    staged_trees.clear();
    write_file(index_file, "");
    // Restore files from commit
    std::string content = read_file(objects_dir + "/" + commit_hash);
    std::string tree_hash;
    std::stringstream ss(content);
    std::string line;
    while (std::getline(ss, line)) {
        if (line.find("tree: ") == 0) {
            tree_hash = line.substr(6);
            break;
        }
    }
    std::unordered_map<std::string, std::string> files;
    collect_files(tree_hash, "", files);
    for (auto& [path, hash] : files) {
        std::string file_content = read_file(objects_dir + "/" + hash);
        write_file(path, file_content);
        staging_area[path] = hash;
        build_tree_hierarchy(path, hash);
    }
// updating file of index
    std::stringstream index;
    for (auto& [f, h] : staging_area) index << f << " " << h << "\n";
    write_file(index_file, index.str());
```

```
    std::cout << "Checked out " << target << "\n";
}
// finds lowest common ancestor of 2 commits by using depth search method


std::string find_lca(const std::string& c1, const std::string& c2) {
    std::unordered_set<std::string> ancestors;
    std::function<void(const std::string&)> collect = [&](const std::string& commit) {
        std::string current = commit;
        while (!current.empty()) {
            ancestors.insert(current);
            std::string content = read_file(objects_dir + "/" + current);
            std::stringstream ss(content);
            std::string line, parent;
            while (std::getline(ss, line)) {
                if (line.find("parent: ") == 0) {
                    parent = line.substr(8);
                    break;
                }
            }
            current = parent;
        }
    };
    collect(c1);
    std::string current = c2;
    while (!current.empty()) {
        if (ancestors.count(current)) return current;
        std::string content = read_file(objects_dir + "/" + current);
        std::stringstream ss(content);
        std::string line, parent;
        while (std::getline(ss, line)) {
            if (line.find("parent: ") == 0) {
                parent = line.substr(8);
                break;
            }
        }
        current = parent;
    }
    return "";  // if there is no common ancestor being found
}


void merge(const std::string& branch) {
    if (!fs::exists(refs_dir + "/" + branch)) {
        std::cout << "Error: Branch not found\n";
        return;
    }
    std::string head = read_file(head_file).substr(5);
```

```
        std::string c1 = read_file(refs_dir + "/" + head);
        std::string c2 = read_file(refs_dir + "/" + branch);
        std::string lca = find_lca(c1, c2);
        if (lca.empty()) {
            std::cout << "Error: No common ancestor\n";
            return;
        }
    // processing file trees for commits
        std::unordered_map<std::string, std::string> files1, files2, files_lca;
        std::function<void(const std::string&, std::unordered_map<std::string, std::string>&)>
load = [&](const std::string& commit, auto& files) {
            std::string content = read_file(objects_dir + "/" + commit);
            std::string tree_hash;
            std::stringstream ss(content);
            std::string line;
            while (std::getline(ss, line)) {
                if (line.find("tree: ") == 0) {
                    tree_hash = line.substr(6);
                    break;
                }
            }
            collect_files(tree_hash, "", files);
        };
        load(c1, files1);
        load(c2, files2);
        load(lca, files_lca);
        bool conflict = false;
    //Compare files and head to apply changes
        for (auto& [path, hash2] : files2) {
            auto it1 = files1.find(path);
            auto it_lca = files_lca.find(path);
            std::string hash1 = it1 != files1.end() ? it1->second : "";
            std::string hash_lca = it_lca != files_lca.end() ? it_lca->second : "";
            if (hash1 != hash_lca && hash2 != hash_lca && hash1 != hash2) {
                std::cout << "CONFLICT: both modified " << path << "\n";
                conflict = true;
                continue;
            }
            if (hash2 != hash_lca) {
                std::string content = read_file(objects_dir + "/" + hash2);
                write_file(path, content);
                staging_area[path] = hash2;
                build_tree_hierarchy(path, hash2);
            }
        }
        if (conflict) {
            std::cout << "Merge failed due to conflicts\n";
            return;
```

```
    }
// create merge commit
    build_tree_hierarchy("");
    std::string tree_hash = create_tree(".");
    std::stringstream commit;
    std::string timestamp = get_time();
    commit << "tree: " << tree_hash << "\n";
    commit << "parent: " << c1 << "," << c2 << "\n";
    commit << "timestamp: " << timestamp << "\n";
    commit << "message: Merge branch '" << branch << "'\n";
    std::string hash = compute_sha1(commit.str());
    write_file(objects_dir + "/" + hash, commit.str());
    write_file(refs_dir + "/" + head, hash);
    staging_area.clear();
    staged_trees.clear();
    write_file(index_file, "");
    std::cout << "Merged " << branch << "\n";
}
```