Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Fall, Year: 2025), B.Sc. in CSE (Day)

Duplicate File Finder and Cleaner

Course Title: Operating Systems Lab
Course Code: CSE-402
Section: 231 D3

Students Details

Name	ID
Md Rahul Shaikh	231902055
Md Mahabub Hasan Mahin	231902056

Submission Date: 16-12-25
Course Teacher's Name: Md. Solaiman Mia

[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

The Duplicate File Finder and Cleaner is a command-line based software tool developed using Bash scripting for Unix-based operating systems such as Linux and macOS. The main purpose of this project is to detect duplicate files by checking their actual content instead of only file names. It also finds empty files and empty folders that remain unused in the system. By removing these unnecessary files, the tool helps improve system performance and saves valuable storage space.

With the rapid increase in digital data, users store a large number of files on their computers every day. Files are copied, moved, and downloaded many times, which creates duplicate copies without the user noticing. These duplicate files waste disk space and slow down file searches. This project provides a lightweight, fast, and easy solution to manage files efficiently without installing extra software. Duplicate file detection and file system management are common operating system tasks [1].

1.2 Motivation

In daily computer usage, people often download the same files more than once or copy files into different folders. Over time, this creates many duplicate files across the system. Managing these files manually becomes difficult, especially when file names are changed. As a result, users face storage problems and confusion while organizing their data.

Another common issue is the presence of empty files and folders. These are usually created because of incomplete downloads, software crashes, or failed installations. Most existing tools either ignore empty files or are difficult to use. This motivated the development of a simple Bash-based tool that works directly from the terminal and shows clear and understandable output.

1.3 Problem Definition

1.3.1 Problem Statement

Manually identifying duplicate files is a slow and unreliable process. Two files may have different names or timestamps but still contain the same data. Because of this, users cannot depend on file names alone to detect duplicates. Empty files and folders are also hard to notice without special tools, especially in large directories.

Many available software tools are either complex, require installation, or do not provide combined detection of duplicate and empty files. Some tools also delete files automatically without proper user confirmation. Therefore, there is a strong need for a simple, automated, and user-friendly tool that can clean file systems safely and efficiently.

1.3.2 Complex Engineering Problem

This project addresses a complex engineering problem related to file system analysis and data management. The system must scan a large number of files and compare them accurately without consuming too much memory or time. It also needs to interact with the user to avoid accidental deletion of important files.

Ensuring both performance and safety at the same time is challenging. The tool must balance speed and accuracy while handling different file sizes and structures. Proper design is required so that scanning, hashing, grouping, and cleaning work together smoothly.

Table 1.1: Summary of the attributes touched by the mentioned projects

Name of the P Attributess	**Explain how to address**
P1: Depth of knowledge required	The project needs basic to moderate knowledge of Bash scripting, file systems, and Operating Systems concepts.
P2: Range of conflicting requirements	—-
P3: Depth of analysis required	File content and hash values must be analysed carefully to identify duplicate files correctly.
P4: Familiarity of issues	—-
P5: Extent of applicable codes	Standard Bash commands like find, md5sum, sort, and stat are used in this project.
P6: Extent of stakeholder involvement and conflicting requirements	—-
P7: Interdependence	Each system step depends on the previous step, such as scanning before hashing and cleaning.

1.4 Design Goals and Objectives

The design of this project focuses on simplicity and reliability. The tool should be easy to use even for users with basic command-line knowledge. It should give clear results so that users can understand which files are safe to delete.

The main objectives of the project are listed below:

- Scan directories **quickly** and **efficiently**
- Detect duplicate files using **MD5** hashing
- **Identify** empty files and empty folders
- Display results in a clear and readable format
- Ensure safe deletion with user confirmation

1.5 Application

The Duplicate File Finder and Cleaner can be used in many real-life situations. Students can use it to clean their study materials and project folders. Developers can use it to manage source code directories and backup files.

It is also useful in computer labs, offices, and server environments where many users store data. By removing duplicate and empty files, the tool helps maintain an organized and efficient file system.

Chapter 2

Design, Development and Implementation

2.1 Introduction

This chapter explains the design, development, and implementation of the Duplicate File Finder and Cleaner system. The main goal of the design is to keep the system simple, reliable, and safe for users. Bash scripting is used so that the tool can run on most Unix-based operating systems without installing extra software.

The system follows a step-by-step development process. Each task such as scanning, duplicate detection, and cleaning is handled separately. This modular approach makes the system easy to understand, test, and improve in the future.

2.2 System Design

The system is designed as a command-line based application. The user provides a directory path as input, and the system scans all files and folders inside that directory. File analysis is done based on file content rather than file names to ensure accurate results.

The design avoids complex graphical interfaces and focuses on simple text output. Important messages are shown clearly so that users can easily understand the results. The system never deletes files automatically, which ensures user safety.

2.3 Project Modules

The project is divided into several functional modules. Each module performs a specific task and passes its output to the next module. This structure improves system reliability and reduces complexity.

The main modules of the system are:

- Directory scanning module
- Empty file and folder detection module
- Hash generation module
- Duplicate file grouping module
- Cleaning and deletion module

2.3.1 Directory Scanning Module

This module scans the given directory and all subdirectories using standard Bash commands. It creates a list of all files and folders present in the system. The scanning process is efficient and works well for medium-sized directories.

This module is the starting point of the system. Accurate scanning is important because all other modules depend on this step.

2.3.2 Empty File and Folder Detection Module

After scanning, the system checks for empty files and empty folders. Empty files are detected by checking if the file **size is zero**. Empty folders are identified if they contain no files or subfolders.

This module helps remove useless data and improves file system organization. It also reduces unnecessary clutter in directories.

2.3.3 Hash Generation Module

In this module, MD5 hash values are generated for all non-empty files. A hash value represents the actual content of a file. Files that have the same hash value and file size are considered duplicate files.

Hash-based comparison is more reliable than name-based comparison because it detects duplicates based on file content rather than file names or locations. This method ensures accurate duplicate file detection even when files are renamed or stored in different directories [6].

2.3.4 Duplicate File Grouping Module

After generating hash values, files are grouped based on matching hash values and file sizes. Each group contains files with identical content. One file is treated as the original, and the remaining files are marked as duplicates.

The grouped results are displayed clearly so that users can review duplicate files before cleaning.

2.3.5 Cleaning and Deletion Module

This module handles the removal of duplicate and empty files. Before deleting any file, the system asks for user confirmation. This prevents accidental deletion of important data.

Only files approved by the user are deleted. This approach ensures safety and builds user trust in the system.

2.4 Flowchart of the System

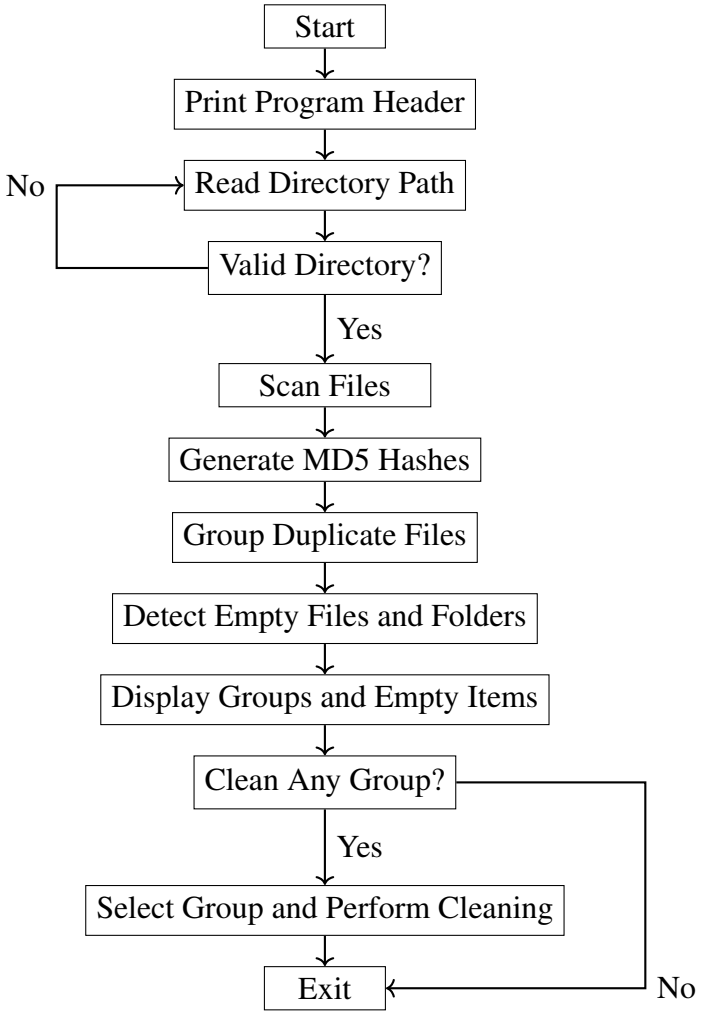

Figure 2.1: Flowchart of the Duplicate File Finder and Cleaner System

2.5 Implementation Details

The system is implemented entirely using Bash scripting. Bash is chosen because it is lightweight, fast, and available by default on most Unix-based operating systems.

Standard Unix utilities are used to perform directory scanning, file analysis, and result organization [3, 4].

All file operations such as scanning, hashing, and sorting are performed using built-in command-line tools. This makes the system portable and easy to run without installing any additional software. The implementation focuses on simplicity, accuracy, and user safety.

2.5.1 Tools and Commands Used

The main tools and commands used in this project are listed below:

- `find` – used for scanning directories and files
- `stat` – used for checking file size
- `md5sum` – used for generating hash values
- `sort` – used for organizing hash results
- Bash loops and conditional statements – used for program control

These command-line tools follow standard UNIX design principles and are widely used for system-level file management tasks [5, 8].

2.6 Algorithm Description

This section describes the algorithm used for detecting duplicate files and empty items. The algorithm follows a step-by-step approach to ensure accurate results and safe file cleaning.

2.6.1 Algorithm for Duplicate File Detection

Input: Directory Path
Output: Duplicate file groups

1. Read directory path from the user
2. Scan all files and folders in the directory
3. Identify empty files and empty folders
4. Generate MD5 hash values for non-empty files
5. Compare file sizes and hash values
6. Group files with identical content
7. Ask user confirmation before cleaning

8. Delete selected files safely

This algorithm ensures accurate duplicate file detection while maintaining user control and system reliability. User confirmation before deletion prevents accidental data loss and improves overall safety.

Chapter 3

Performance Evaluation

3.1 Introduction

This chapter evaluates the performance of the Duplicate File Finder and Cleaner system. The main objective of this evaluation is to verify the correctness, efficiency, and reliability of the system during real execution. The system is tested using different directories containing duplicate files, empty files, and empty folders.

The evaluation focuses on scanning accuracy, duplicate detection, empty file detection, and safe cleaning behavior. Real execution outputs are captured and presented to demonstrate the working of the system in practical scenarios.

3.2 Simulation Environment and Simulation Procedure

3.2.1 Simulation Environment

The system is tested in a practical and controlled environment that represents a typical user system. The testing setup ensures that the results are realistic and applicable to everyday usage.

The simulation environment is summarized below:

- Operating System: Linux
- Shell Environment: Bash
- Hardware: Standard laboratory computer
- Storage Type: Local file system

3.2.2 Simulation Procedure

First, the program is executed from the terminal. The user provides a directory path for scanning. The selected directories contain different types of files, including duplicate files, empty files, and empty folders.

Next, the system scans all files and folders inside the directory. During scanning, a progress bar is displayed to show scanning status. MD5 hash values are generated for non-empty files, and empty files and folders are identified.

After scanning, duplicate file groups and empty items are displayed. The user is then asked whether they want to clean any group. If the user chooses to clean, the system deletes duplicate or empty files safely after confirmation. This process continues until no duplicate or empty items remain.

3.3 Execution Results

This section presents the execution results of the system using real test cases. The following figures show different stages of system execution, including directory scanning, duplicate file detection, and file cleaning.

3.3.1 Directory Scanning with No Duplicate Files

Figure 3.1 shows the execution result when no duplicate files or empty items are found in the selected directory. The system completes the scanning process successfully and exits safely without performing any cleaning.

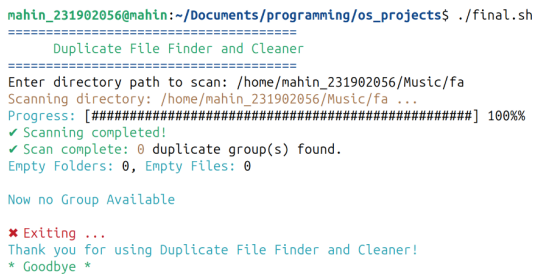

Figure 3.1: Execution Output Showing No Duplicate or Empty Files

3.3.2 Execution Output: Detection of Duplicate Files

Figure 3.2 shows the system scanning the selected directory and detecting duplicate files successfully. The output displays the original file, its size, and the list of duplicate copies based on file content. The system supports cleaning one group at a time and does not support deleting multiple groups simultaneously.

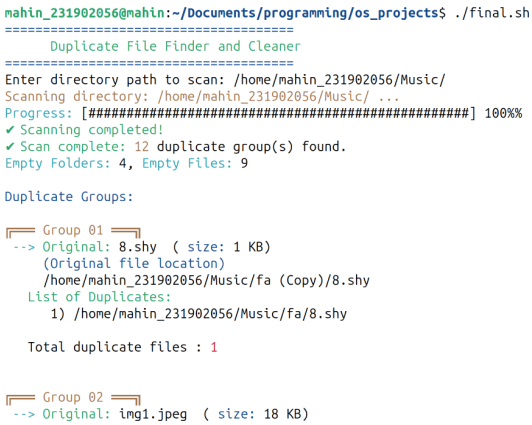

Figure 3.2: Terminal Output Showing Detection of Duplicate Files

3.3.3 Cleaning Empty Files

Figure 3.3 shows the detection and cleaning of empty files. The system lists empty files as a group and deletes them only after user confirmation. This confirms safe and controlled cleaning behavior.

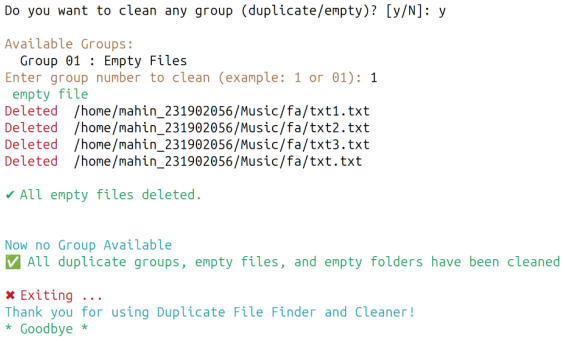

Figure 3.3: Detection and Cleaning of Empty Files

3.3.4 Cleaning Duplicate Image Files

Figure 3.4 shows duplicate image file detection and cleaning. The system keeps one original file and deletes all duplicate copies. The output clearly displays deleted file paths and confirms successful cleaning.

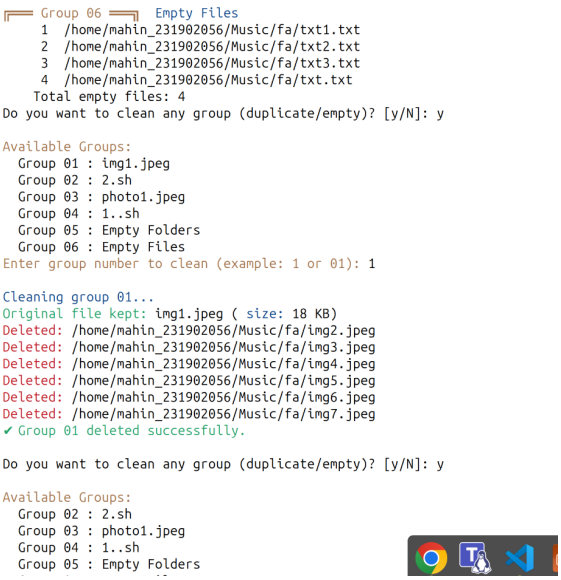

Figure 3.4: Detection and Cleaning of Duplicate Image Files

3.4 Results Analysis

The execution results confirm that the system accurately detects duplicate files based on file content using MD5 hash values. Files with the same content but different names or locations are grouped correctly.

Empty files and empty folders are also detected correctly. The system provides clear output and summary information, making it easy for users to understand the results. No files are deleted automatically, ensuring full user control and safety.

The observed results confirm the effectiveness of hash-based duplicate detection techniques in file system maintenance [1].

3.5 Performance Metrics

The performance of the system is evaluated using the following metrics:

- Accuracy of duplicate file detection
- Correct identification of empty files and folders
- Execution time during scanning
- Memory usage during execution
- Safety during file deletion

The system performs efficiently for medium-sized directories and maintains low memory usage.

3.6 Discussion of Results

The results demonstrate that Bash scripting is effective for file system maintenance tasks. The use of standard Unix commands ensures fast execution and good portability. Hash-based comparison improves detection accuracy.

The interactive cleaning process prevents accidental data loss and increases user confidence. Overall, the system meets its design objectives and performs reliably in real-world scenarios.

3.7 Complex Engineering Problem Discussion

The project addresses a complex engineering problem involving file system scanning, data comparison, and safe deletion. Managing large numbers of files while maintaining accuracy and safety is challenging.

The modular design of the system helps manage this complexity. Each module works together smoothly, ensuring correct and efficient system operation.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

This project successfully developed a Duplicate File Finder and Cleaner system using Bash scripting. The main goal of the system was to detect duplicate files, empty files, and empty folders and clean them safely. The system performs file analysis based on actual file content, which ensures accurate duplicate detection.

The system provides clear and interactive output through the command-line interface. Users are always asked for confirmation before any file deletion, which prevents accidental data loss. The use of standard Unix tools makes the system lightweight, fast, and easy to use on most Unix-based operating systems.

Performance evaluation results show that the system works reliably for different directories. It accurately detects duplicate file groups and empty items and cleans them successfully when requested by the user. Overall, the system meets its design objectives and solves a common real-world file management problem.

4.2 Limitations

The Duplicate File Finder and Cleaner system works effectively for detecting and cleaning duplicate and empty files. However, like any command-line based tool, it has some functional and usability limitations.

- The system supports cleaning only one group at a time and does not support deleting multiple groups simultaneously.
- The tool works only on local file systems and does not support cloud or network storage.
- MD5 hashing is used for file comparison, which is not suitable for security-critical applications.
- The system is command-line based, which may be difficult for users unfamiliar with terminal usage.

- Performance may decrease when scanning very large directories with a large number of files.

4.3 Scope of Future Work

The Duplicate File Finder and Cleaner system can be improved further by adding new features and enhancing usability. Future development can make the tool more flexible, faster, and easier to use for a wider range of users.

- Support for cleaning multiple duplicate groups at the same time.
- Development of a graphical user interface (GUI) for easier interaction.
- Use of stronger hashing algorithms such as SHA-256 for improved reliability.
- Support for cloud storage and network-based file systems.
- Performance optimization for very large directories.
- Addition of scan history, logging, and report generation features.

References

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 9th ed. Hoboken, NJ, USA: Wiley, 2018.

[2] W. E. Shotts, *The Linux Command Line: A Complete Introduction*. San Francisco, CA, USA: No Starch Press, 2019.

[3] Free Software Foundation, *GNU Bash Reference Manual*. [Online]. Available: https://www.gnu.org/software/bash/manual/. Accessed: 2025.

[4] Free Software Foundation, *GNU Findutils Manual*. [Online]. Available: https://www.gnu.org/software/findutils/. Accessed: 2025.

[5] Free Software Foundation, *GNU Coreutils Manual*. [Online]. Available: https://www.gnu.org/software/coreutils/. Accessed: 2025.

[6] R. L. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321, Internet Engineering Task Force (IETF), 1992.

[7] Linux Documentation Project, *Filesystem Hierarchy Standard (FHS)*. [Online]. Available: https://refspecs.linuxfoundation.org/fhs.shtml. Accessed: 2025.

[8] The Open Group, *UNIX Command-Line Utilities*. [Online]. Available: https://pubs.opengroup.org/. Accessed: 2025.