universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Sistemas Operativos **Ano lectivo 2025/2026**

Trabalho Prático 1

Linux Recycle Bin Simulation – Complete Project Proposal[1]

Table of Contents

---

1. Project Overview

1.1 Introduction

You will develop a complete **Linux Recycle Bin System** using Bash shell scripting. This system will replicate the functionality of the Windows Recycle Bin, allowing users to safely delete files with the ability to restore them before permanent deletion.

1.2 Project Context

- **Course:** Operating Systems
- **Team Size:** Two students project

[1] The text of this project proposal had AI contributions to its completion.

- **Programming Language:** Bash Shell Script (version 4.0 or higher)

1.3 Real-World Application

This project simulates real system utilities and teaches you:

- Safe file management practices
- Metadata tracking and preservation
- User data protection mechanisms
- System programming fundamentals
- Professional code organization

---

2. Learning Objectives

Upon successful completion, you will demonstrate ability to:

✓ Write modular shell scripts using functions
✓ Implement safe file system operations
✓ Manage metadata and persistent storage
✓ Handle errors gracefully and validate user input
✓ Create intuitive command-line interfaces
✓ Apply Unix/Linux file permissions and ownership concepts
✓ Debug and test shell scripts systematically
✓ Document code professionally

---

3. Technical Requirements

3.1 System Requirements

- **Operating System:** Linux (Ubuntu 24.04, Fedora, or similar)
- **Shell:** Bash 4.0 or higher
- **Required Commands:** `mv`, `rm`, `cp`, `stat`, `date`, `grep`, `awk`, `sed`
- **Optional Tools:** `shellcheck` (for validation), `git` (for version control)

3.2 Project Structure

Your project must include:

```
StudentName_RecycleBin/
├── recycle_bin.sh           # Main executable script
├── README.md                # User documentation
├── TECHNICAL_DOC.md         # Technical documentation
├── TESTING.md               # Test documentation
├── test_suite.sh            # Automated test script
```

```
└── screenshots/              # Directory with demo screenshots
    ├── delete_operation.png
    ├── list_view.png
    ├── restore_operation.png
    └── stats_view.png
```

3.3 Recycle Bin Architecture

Directory Structure

```
~/.recycle_bin/
├── files/              # Stores deleted files with unique IDs
├── metadata.db         # CSV database of file information
├── config              # Configuration file
└── recyclebin.log      # Operation log file
```

Metadata Schema (CSV Format)

```
ID,ORIGINAL_NAME,ORIGINAL_PATH,DELETION_DATE,FILE_SIZE,FILE_TYPE,PERMIS-
SIONS,OWNER
1696234567_abc123,document.txt,/home/user/Documents/document.txt,2024-10-
02 14:30:22,4096,file,644,user:user
```

Field Descriptions:

- **ID:** Unique identifier (timestamp_randomstring format)
- **ORIGINAL_NAME:** Original filename or directory name
- **ORIGINAL_PATH:** Complete absolute path of original location
- **DELETION_DATE:** Timestamp when deleted (YYYY-MM-DD HH:MM:SS)
- **FILE_SIZE:** Size in bytes
- **FILE_TYPE:** Either "file" or "directory"
- **PERMISSIONS:** Original permission bits (e.g., 644, 755)
- **OWNER:** Original owner and group (user:group format)

---

4. Implementation Specifications

4.1 Mandatory Features (Core Requirements)

Feature 1: Initialize Recycle Bin

Function Name: `initialize_recyclebin()`

Requirements:

- Create `~/.recycle_bin/` directory structure if not exists
- Create subdirectory `files/` for storing deleted items
- Initialize `metadata.db` with CSV header
- Create default `config` file with settings
- Create empty `recyclebin.log` file

Configuration File Format:

```
MAX_SIZE_MB=1024
```

```
RETENTION_DAYS=30
```

Feature 2: Delete Files/Directories

Function Name: `delete_file()`

Requirements:

- Accept one or more file/directory paths as arguments
- Validate that files exist before deletion
- Generate unique ID for each deleted item
- Move files to `~/.recycle_bin/files/` with unique ID as filename
- Extract and store metadata:
    - Original filename and absolute path
    - Deletion timestamp
    - File size (use `stat` or `du` commands)
    - File type (file or directory)
    - Original permissions (using `stat -c %a`)
    - Original owner (using `stat -c %U:%G`)
- Append metadata entry to `metadata.db`
- Provide user feedback (success/failure messages)
- Support recursive deletion for directories
- Log all operations to `recyclebin.log`

Error Handling:

- File doesn't exist
- No read/write permissions
- Insufficient disk space
- Cannot delete recycle bin itself

Example Usage:

```
./recycle_bin.sh delete myfile.txt
./recycle_bin.sh delete file1.txt file2.txt directory/
```

Feature 3: List Recycle Bin Contents

Function Name: `list_recycled()`

Requirements:

- Display all items currently in recycle bin
- Show in formatted table with columns:
    - Unique ID (truncated for display)
    - Original filename
    - Deletion date and time
    - File size (human-readable format: B, KB, MB, GB)
- Implement two view modes:
    - **Normal mode:** Compact table view

    - **Detailed mode:** Full information per item (using `--detailed` flag)
- Display total item count
- Display total storage used
- Handle empty recycle bin gracefully

Example Usage:

```
./recycle_bin.sh list
./recycle_bin.sh list --detailed
```

Feature 4: Restore Files

Function Name: `restore_file()`

Requirements:

- Accept file ID or filename as parameter
- Search metadata for matching entry
- Restore file to original absolute path
- Restore original permissions using `chmod`
- Remove entry from metadata.db after successful restoration
- Handle restoration conflicts:
    - If original path no longer exists, create parent directories
    - If file already exists at original path, ask user for action:
        - Overwrite existing file
        - Restore with modified name (append timestamp)
        - Cancel operation
- Provide restoration feedback
- Log restoration operations

Error Handling:

- File ID not found
- Original directory no longer exists
- Permission denied at destination
- Disk space issues

Example Usage:

```
./recycle_bin.sh restore 1696234567_abc123
./recycle_bin.sh restore myfile.txt
```

Feature 5: Search Files

Function Name: `search_recycled()`

Requirements:

- Accept search pattern as parameter
- Search both filename and original path

- Support wildcard patterns (e.g., "*.txt", "*report*")
- Display matching results in table format
- Show message if no matches found
- Case-insensitive search option

Example Usage:

```
./recycle_bin.sh search "report"
./recycle_bin.sh search "*.pdf"
```

Feature 6: Empty Recycle Bin

Function Name: `empty_recyclebin()`

Requirements:

- Support two modes:
  - **Empty all:** Permanently delete all items
  - **Empty specific:** Delete single item by ID
- Require user confirmation before permanent deletion
- Provide `--force` flag to skip confirmation (dangerous!)
- Permanently delete files using `rm -rf`
- Update metadata.db accordingly
- Display summary of deleted items
- Log deletion operations

Example Usage:

```
./recycle_bin.sh empty
./recycle_bin.sh empty 1696234567_abc123
./recycle_bin.sh empty --force
```

Feature 7: Help System

Function Name: `display_help()`

Requirements:

- Display comprehensive usage information
- Show all available commands with descriptions
- Provide usage examples
- Document all command-line options
- Show configuration file location

Example Usage:

```
./recycle_bin.sh help
./recycle_bin.sh --help
./recycle_bin.sh -h
```

4.2 Optional Features (Extra Credit)

Feature 8: Statistics Dashboard (10 points)

Function Name: `show_statistics()`

Requirements:

- Display total number of items in recycle bin
- Show total storage used with quota percentage
- Break down by file type (files vs directories)
- Show oldest and newest items
- Display average file size

Feature 9: Auto-Cleanup (10 points)

Function Name: `auto_cleanup()`

Requirements:

- Automatically delete items older than RETENTION_DAYS
- Read retention period from config file
- Provide cleanup summary
- Run manually or integrate into delete operation

Feature 10: Quota Management (5 points)

Function Name: `check_quota()`

Requirements:

- Check if recycle bin exceeds MAX_SIZE_MB
- Display warning when quota exceeded
- Optionally trigger auto-cleanup when full

Feature 11: File Preview (5 points)

Function Name: `preview_file()`

Requirements:

- Show first 10 lines for text files
- Display file type information for binary files
- Accept file ID as parameter

5. Development Guidelines

5.1 Code Organization

Your script must follow this structure:

```
#!/bin/bash

#################################################
# Script Header Comment
# Author: Your Name
# Date: YYYY-MM-DD
# Description: Brief description
# Version: 1.0
#################################################

# Global Variables (ALL CAPS)
RECYCLE_BIN_DIR="$HOME/.recycle_bin"
METADATA_FILE="$RECYCLE_BIN_DIR/metadata.db"

# Color Codes (optional but recommended)
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m'

# Function Definitions
# - Each function must have a header comment
# - Use descriptive function names
# - Validate all parameters
# - Return appropriate exit codes

function_name() {
    # Function body
}

# Main Program Logic
main() {
    # Initialize
    # Parse arguments
    # Execute commands
}

# Script Entry Point
main "$@"
```

5.2 Coding Standards

Variable Naming

- **Global variables:** UPPERCASE with underscores (e.g., `RECYCLE_BIN_DIR`)
- **Local variables:** lowercase with underscores (e.g., `file_path`)
- **Function names:** lowercase with underscores (e.g., `delete_file`)

Function Comments

Each function must have a header comment:

```
##################################################
# Function: function_name
# Description: What the function does
# Parameters: $1 - description, $2 - description
# Returns: 0 on success, 1 on failure
##################################################
```

Error Handling

- Check return codes of all commands
- Use meaningful error messages
- Always validate user input
- Handle edge cases (empty strings, special characters, etc.)

Quoting

- Always quote variables: `"$variable"`
- Use quotes for paths: `"$file_path"`
- Prevents word splitting and globbing issues

5.3 Best Practices

1. **Use `set -e`**: Exit on error (optional but recommended)
2. **Validate inputs**: Check arguments before processing
3. **Provide feedback**: Inform user of operations being performed
4. **Log operations**: Write important events to log file
5. **Use functions**: Break code into reusable modules
6. **Handle signals**: Trap SIGINT/SIGTERM for cleanup
7. **Avoid hardcoding**: Use variables for paths and settings
8. **Test incrementally**: Test each function as you develop

5.4 Security Considerations

⚠ **IMPORTANT SECURITY RULES:**

1. **Never execute unvalidated input**
2. **Sanitize file paths** (check for `..`, absolute paths, etc.)
3. **Validate file operations** before execution
4. **Don't follow symbolic links** that could escape recycle bin
5. **Respect file permissions** - don't force operations
6. **Prevent recycle bin deletion** - block self-reference
7. **Use safe temporary files** with `mktemp`

6. Testing Requirements

6.1 Test Categories

You must test all of the following scenarios:

Basic Functionality Tests

- [ ] Initialize recycle bin structure
- [ ] Delete single file
- [ ] Delete multiple files in one command
- [ ] Delete empty directory
- [ ] Delete directory with contents (recursive)
- [ ] List empty recycle bin
- [ ] List recycle bin with items
- [ ] Restore single file
- [ ] Restore to non-existent original path
- [ ] Empty entire recycle bin
- [ ] Search for existing file
- [ ] Search for non-existent file
- [ ] Display help information

Edge Cases

- [ ] Delete non-existent file
- [ ] Delete file without permissions
- [ ] Restore when original location has same filename
- [ ] Restore with ID that doesn't exist
- [ ] Handle filenames with spaces
- [ ] Handle filenames with special characters (`!@#$%^&*()`)
- [ ] Handle very long filenames (255+ characters)
- [ ] Handle very large files (>100MB)
- [ ] Handle symbolic links
- [ ] Handle hidden files (starting with `.`)
- [ ] Delete files from different directories
- [ ] Restore files to read-only directories

Error Handling

- [ ] Invalid command line arguments
- [ ] Missing required parameters
- [ ] Corrupted metadata file
- [ ] Insufficient disk space
- [ ] Permission denied errors
- [ ] Attempting to delete recycle bin itself
- [ ] Concurrent operations (run two instances)

Performance Tests

- [ ] Delete 100+ files
- [ ] List recycle bin with 100+ items
- [ ] Search in large metadata file
- [ ] Restore from bin with many items

6.2 Automated Test Script

Create `test_suite.sh` with automated tests:

```
#!/bin/bash

# Test Suite for Recycle Bin System

SCRIPT="./recycle_bin.sh"
TEST_DIR="test_data"
PASS=0
FAIL=0

# Colors
GREEN='\033[0;32m'
RED='\033[0;31m'
NC='\033[0m'

# Test Helper Functions
setup() {
    mkdir -p "$TEST_DIR"
    rm -rf ~/.recycle_bin
}

teardown() {
    rm -rf "$TEST_DIR"
    rm -rf ~/.recycle_bin
}

assert_success() {
    if [ $? -eq 0 ]; then
        echo -e "${GREEN}✓ PASS${NC}: $1"
        ((PASS++))
    else
        echo -e "${RED}✗ FAIL${NC}: $1"
        ((FAIL++))
    fi
}

assert_fail() {
    if [ $? -ne 0 ]; then
        echo -e "${GREEN}✓ PASS${NC}: $1"
        ((PASS++))
    else
        echo -e "${RED}✗ FAIL${NC}: $1"
        ((FAIL++))
    fi
}
```

```
# Test Cases
test_initialization() {
    echo "=== Test: Initialization ==="
    setup
    $SCRIPT help > /dev/null
    assert_success "Initialize recycle bin"
    [ -d ~/.recycle_bin ] && echo "✓ Directory created"
    [ -f ~/.recycle_bin/metadata.db ] && echo "✓ Metadata file created"
}

test_delete_file() {
    echo "=== Test: Delete File ==="
    setup
    echo "test content" > "$TEST_DIR/test.txt"
    $SCRIPT delete "$TEST_DIR/test.txt"
    assert_success "Delete existing file"
    [ ! -f "$TEST_DIR/test.txt" ] && echo "✓ File removed from original
location"
}

test_list_empty() {
    echo "=== Test: List Empty Bin ==="
    setup
    $SCRIPT list | grep -q "empty"
    assert_success "List empty recycle bin"
}

test_restore_file() {
    echo "=== Test: Restore File ==="
    setup
    echo "test" > "$TEST_DIR/restore_test.txt"
    $SCRIPT delete "$TEST_DIR/restore_test.txt"

    # Get file ID from list
    ID=$($SCRIPT list | grep "restore_test" | awk '{print $1}')
    $SCRIPT restore "$ID"
    assert_success "Restore file"
    [ -f "$TEST_DIR/restore_test.txt" ] && echo "✓ File restored"
}

# Run all tests
echo "========================================="
echo "  Recycle Bin Test Suite"
echo "========================================="

test_initialization
test_delete_file
test_list_empty
test_restore_file

# Add more test functions here

teardown

echo "========================================="
echo "Results: $PASS passed, $FAIL failed"
echo "========================================="
```

```
[ $FAIL -eq 0 ] && exit 0 || exit 1
```

6.3 Manual Testing Guide

Create a `TESTING.md` document with:

1. **Test scenario description**
2. **Steps to reproduce**
3. **Expected outcome**
4. **Actual outcome**
5. **Screenshots** (when applicable)
6. **Pass/Fail status**

Example format:

```
### Test Case 1: Delete Single File

**Objective:** Verify that a single file can be deleted successfully

**Steps:**
1. Create test file: `echo "test" > test.txt`
2. Run: `./recycle_bin.sh delete test.txt`
3. Verify file is removed from current directory
4. Run: `./recycle_bin.sh list`
5. Verify file appears in recycle bin

**Expected Result:**
- File is moved to ~/.recycle_bin/files/
- Metadata entry is created
- Success message is displayed
- File appears in list output

**Actual Result:** [Fill in after testing]

**Status:** ☐ Pass ☐ Fail

**Screenshots:** [If applicable]
```

---

7. Deliverables

7.1 Source Code

- **recycle_bin.sh** - Main executable script (must be executable: `chmod +x`)
- Well-commented code following style guidelines
- All mandatory features implemented
- Optional features clearly marked

7.2 Documentation

README.md

Must include:

```
# Linux Recycle Bin System

## Author
[Your Name]
[Your Student ID]

## Description
[Brief project description]

## Installation
[How to install/setup]

## Usage
[How to use with examples]

## Features
- [List of implemented features]
- [Mark optional features]

## Configuration
[How to configure settings]

## Examples
[Detailed usage examples with screenshots]

## Known Issues
[Any limitations or bugs]

## References
[Resources used]
```

TECHNICAL_DOC.md

Must include:

- System architecture diagram (ASCII art or image)
- Data flow diagrams
- Metadata schema explanation
- Function descriptions
- Design decisions and rationale
- Algorithm explanations
- Flowcharts for complex operations

TESTING.md

Must include:

- Test plan overview
- Test cases with results

- Edge cases tested
- Known bugs or limitations
- Test coverage summary

7.3 Demonstration Materials

- **Screenshots folder** with at least 5 screenshots showing:
  - Delete operation
  - List view (normal and detailed)
  - Restore operation
  - Search results
  - Statistics/optional features
- **Optional:** Screen recording (5 minutes) demonstrating all features

7.4 Test Suite

- **test_suite.sh** - Automated test script
- Must test at least 15 scenarios
- Must report pass/fail results
- Must be executable and well-documented

---

8. Evaluation Rubric

Total Points: 100

A. Functionality (40 points)

- [ ] Initialize recycle bin (5 points)
- [ ] Delete files/directories (10 points)
- [ ] List recycle bin contents (8 points)
- [ ] Restore files (10 points)
- [ ] Search functionality (4 points)
- [ ] Empty recycle bin (3 points)

B. Code Quality (25 points)

- [ ] Proper function modularity (7 points)
- [ ] Code comments and documentation (6 points)
- [ ] Error handling (6 points)
- [ ] Code style and conventions (3 points)
- [ ] Variable naming and organization (3 points)

C. Documentation (20 points)

- [ ] README.md completeness (7 points)
- [ ] Technical documentation quality (7 points)
- [ ] Testing documentation (6 points)

D. Testing (10 points)

- [ ] Comprehensive test cases (5 points)
- [ ] Automated test script (3 points)
- [ ] Test coverage and results (2 points)

E. Innovation & Extra Credit (5 points + bonus)

- [ ] User experience enhancements (2 points)
- [ ] Creative solutions (3 points)
- [ ] **Bonus:** Statistics feature (+10 points)
- [ ] **Bonus:** Auto-cleanup feature (+10 points)
- [ ] **Bonus:** Other optional features (+5 points each)

Grading Scale

- **[18;20]:** Excellent - All features work, well-documented, tested
- **[16;18[:** Good - Most features work, adequate documentation
- **[14;16[:** Satisfactory - Core features work, basic documentation
- **[12;14[:** Needs Improvement - Some features work, minimal documentation
- **<12:** Unsatisfactory - Major features missing or broken

---

9. Support Resources

9.1 Office Hours

- **When: Tue, 10am-11am / Thu, 15pm-16pm**
- **Where:** 4.1.01 / 4.2.25
- **What to bring:** Your code, error messages, specific questions

9.2 Discussion Forum

- **Platform:** [eLearning]
- **Guidelines:**
  - Search before posting
  - No complete code sharing
  - Help each other with concepts, not solutions
  - Post error messages and what you've tried

9.3 Online Resources

Shell Scripting Guides

- Advanced Bash-Scripting Guide: https://tldp.org/LDP/abs/html/
- GNU Bash Manual: https://www.gnu.org/software/bash/manual/
- ShellCheck (validation tool): https://www.shellcheck.net/

Command References

- `man bash` - Bash manual
- `man stat` - File status command
- `man date` - Date formatting
- `help` - Built-in commands help

Video Tutorials

- YouTube: "Bash Scripting Tutorial for Beginners"
- YouTube: "Linux File Operations"
- YouTube: "Shell Script Functions"

9.4 Starter Code Template

A basic template is provided in the support resources document. You may use this as a starting point, but you must implement all function logic yourself.

9.5 Sample Metadata Format

```
ID,ORIGINAL_NAME,ORIGINAL_PATH,DELETION_DATE,FILE_SIZE,FILE_TYPE,PERMIS-
SIONS,OWNER
1696234567_abc123,document.txt,/home/user/Documents/document.txt,2024-10-
02 14:30:22,4096,file,644,user:user
1696234890_def456,project_folder,/home/user/Projects/project_folder,2024-
10-02 15:45:10,20480,directory,755,user:user
```

9.6 Useful Shell Commands Cheat Sheet

```
# File Information
stat -c "%n %s %a %U:%G" file      # Name, size, perms, owner
file filename                      # File type
du -sb directory                   # Directory size
realpath file                      # Absolute path
basename /path/to/file             # Filename only
dirname /path/to/file              # Directory only

# Date/Time
date +%s                           # Unix timestamp
date "+%Y-%m-%d %H:%M:%S"          # Formatted date

# String Operations
${var:0:10}                        # Substring (first 10 chars)
${var##*/}                         # Remove path
${var%%.*}                         # Remove extension

# File Tests
[ -e file ]                        # Exists
[ -f file ]                        # Is regular file
[ -d dir ]                         # Is directory
[ -r file ]                        # Is readable
[ -w file ]                        # Is writable

# Reading CSV
while IFS=',' read -r col1 col2 col3; do
```

```
    echo "$col1"
done < file.csv

# Generate random string
cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 8 | head -n 1
```

---

10. Submission Instructions

10.1 File Preparation

1. **Organize your files** according to the required structure
2. **Test your script** on a clean Linux system
3. **Verify all documentation** is complete
4. **Include screenshots** in the appropriate folder
5. **Remove temporary files** (e.g., `*~`, `*.swp`)

10.2 Submission Package

Create a compressed archive:

```
tar -czf SO-2526-T1-Px-Gy-11111-22222.tgz StudentName_RecycleBin/
```

10.3 Submission Checklist

Before submitting, verify:

- [ ] Script is executable (`chmod +x recycle_bin.sh`)
- [ ] All functions are implemented
- [ ] Help command works
- [ ] README.md is complete
- [ ] TECHNICAL_DOC.md is complete
- [ ] TESTING.md is complete with results
- [ ] test_suite.sh runs successfully
- [ ] Screenshots are included
- [ ] No syntax errors (`bash -n recycle_bin.sh`)
- [ ] ShellCheck validation passed (optional but recommended)
- [ ] Your name and student ID are in all documents

10.4 Submission Method

Submit via [Canvas/Moodle/Email]:

- **Filename:** `StudentName_RecycleBin.tar.gz` or `.zip`
- **Deadline:** [TBD by instructor]
- **Late Policy:** [TBD by instructor]

10.5 Demo/Presentation

Be prepared to:

- Demonstrate your script live
- Explain your design decisions
- Walk through your code
- Answer questions about implementation
- Show test results

---

11. Timeline and Milestones

Week 1: Planning & Basic Structure

- [ ] Set up development environment
- [ ] Read all project documentation
- [ ] Design system architecture
- [ ] Implement initialization function
- [ ] Implement delete function (basic version)
- [ ] Start documentation

Deliverable: Working initialization and basic delete

Week 2: Core Features

- [ ] Complete delete function (with metadata)
- [ ] Implement list function
- [ ] Implement restore function
- [ ] Begin error handling
- [ ] Continue documentation

Deliverable: Delete, list, and restore working

Week 3: Advanced Features & Testing

- [ ] Implement search function
- [ ] Implement empty function
- [ ] Complete error handling
- [ ] Create automated test suite
- [ ] Manual testing
- [ ] Bug fixes

Deliverable: All core features complete, tested

Week 4: Polish & Documentation

- [ ] Implement optional features (if desired)
- [ ] Complete all documentation
- [ ] Take screenshots
- [ ] Final testing
- [ ] Code review and cleanup
- [ ] Prepare submission package

Deliverable: Final submission on 31st October, 2025

---

12. Academic Integrity

12.1 What is Allowed

✓ Consulting shell scripting references and manuals
✓ Discussing concepts with classmates
✓ Using AI tools to understand concepts or debug
✓ Searching for solutions to specific syntax problems
✓ Using provided templates and examples

12.2 What is NOT Allowed

✗ Copying code from classmates
✗ Sharing your complete solution with others
✗ Using AI to generate complete functions
✗ Copying entire solutions from online sources
✗ Submitting work that is not your own

12.3 Citation Requirements

If you use:

- External code snippets (>3 lines): Cite the source – IEEE template
- AI assistance: Mention in README which parts were assisted
- Online resources: List in references section – IEEE template

12.4 Consequences

Violations of academic integrity will result in:

- Zero on the assignment (minimum)

---

13. Frequently Asked Questions

Q: Can I use other scripting languages?
A: No, this must be implemented in Bash shell scripting.

Q: Can I use external libraries or tools?
A: Only standard Linux utilities (mv, rm, stat, grep, etc.) are allowed.

Q: What if I can't implement all features?
A: Implement as many as possible. Partial credit is awarded. Document what works and what doesn't.

Q: How do I handle files with spaces in names?
A: Always quote variables: `"$filename"` and `"$path"`.

Q: What should I do if the original path no longer exists?
A: Your script should create the directory structure or ask the user where to restore.

Q: Can I modify the metadata format?
A: Yes, but it must still be CSV and include all required fields. Document changes.

Q: How do I test my script without breaking my system?
A: Test in a separate directory or use a virtual machine. Never test in your home directory.

Q: What if my script has bugs I can't fix?
A: Document known issues in README.md and explain what you tried. Partial credit is available.

Q: Can I work with a partner?
A: No, this is an individual project. Collaboration is limited to concept discussion.

Q: How detailed should my comments be?
A: Every function needs a header comment. Complex logic needs inline comments. Err on the side of more documentation.

Q: Will there be a demo/presentation?
A: [TBD by instructor]

---

14. Tips for Success

💡 **Start Early** - Don't wait until the last week
💡 **Test Frequently** - Test each function as you write it
💡 **Use Version Control** - Git can save you if something breaks
💡 **Read Error Messages** - They usually tell you what's wrong
💡 **Use ShellCheck** - It catches many common mistakes

💡 **Ask Questions** - Use office hours and forums
💡 **Document as You Go** - Don't save documentation for the end
💡 **Keep Backups** - Always have a working version saved
💡 **Read the Manual** - `man bash`, `man stat`, etc. are your friends
💡 **Break It Down** - Tackle one function at a time

---

15. Contact Information

Instructor: Pedro Azevedo Fernandes (paf@ua.pt); Nuno Lau (nunolau@ua.pt)
Course Website: https://elearning.ua.pt/course/view.php?id=4329
Discussion Forum: https://elearning.ua.pt/mod/forum/view.php?id=1634485

---

16. Appendix: Complete Example Session

Here's what a complete working session should look like:

```
# Initial setup
$ chmod +x recycle_bin.sh
$ ./recycle_bin.sh help
# [Help information displayed]

# Create test files
$ mkdir test_files
$ echo "Document 1" > test_files/doc1.txt
$ echo "Document 2" > test_files/doc2.txt
$ mkdir test_files/subfolder
$ echo "Document 3" > test_files/subfolder/doc3.txt

# Delete files
$ ./recycle_bin.sh delete test_files/doc1.txt
✓ Moved to recycle bin: doc1.txt
  ID: 1696234567_abc123

$ ./recycle_bin.sh delete test_files/doc2.txt test_files/subfolder
✓ Moved to recycle bin: doc2.txt
  ID: 1696234890_def456
✓ Moved to recycle bin: subfolder
  ID: 1696235000_ghi789

Summary: 3 succeeded, 0 failed

# List recycle bin
$ ./recycle_bin.sh list
=== Recycle Bin Contents ===

ID                        NAME                           DELETED
SIZE
---------------------------------------------------------------------------
--------------
```

```
1696234567_abc123...     doc1.txt                       2024-10-02
14:30:22  11B
1696234890_def456...     doc2.txt                       2024-10-02
14:31:15  11B
1696235000_ghi789...     subfolder                      2024-10-02
14:31:15  4096B

Total items: 3
Total size: 4118B / 1024MB

# List detailed view
$ ./recycle_bin.sh list --detailed
=== Recycle Bin Contents ===

ID: 1696234567_abc123
Name: doc1.txt
Original Path: /home/user/test_files/doc1.txt
Deleted: 2024-10-02 14:30:22
Size: 11B
Type: file
Permissions: 644
Owner: user:user
----------------------------------------

# Search for files
$ ./recycle_bin.sh search "doc"
=== Search Results for: 'doc' ===

ID                       NAME                           DELETED
---------------------------------------------------------------------------
------
1696234567_abc123...     doc1.txt                       2024-10-02
14:30:22
1696234890_def456...     doc2.txt                       2024-10-02
14:31:15

# Restore a file
$ ./recycle_bin.sh restore 1696234567_abc123
Restoring: doc1.txt
  Original path: /home/user/test_files/doc1.txt
✓ Successfully restored: doc1.txt

$ ls test_files/
doc1.txt

# Check statistics
$ ./recycle_bin.sh stats
=== Recycle Bin Statistics ===

Total Items: 2
  Files: 1
  Directories: 1

Total Size: 4107B
Quota: 4107B / 1024MB
Usage: 0%

Oldest item: 2024-10-02 14:31:15
Newest item: 2024-10-02 14:31:15
```

```
# Empty recycle bin
$ ./recycle_bin.sh empty
⚠ WARNING: This will permanently delete all 2 items
Are you sure? This cannot be undone (y/n): y
✓ Recycle bin emptied (2 items permanently deleted)

$ ./recycle_bin.sh list
Recycle bin is empty
```

---

17. Appendix: Common Errors and Solutions

Error 1: Permission Denied

```
$ ./recycle_bin.sh delete /etc/hosts
✗ Error: Permission denied
```

Solution: Don't try to delete system files. Test with your own files.

Error 2: File Not Found

```
$ ./recycle_bin.sh delete nonexistent.txt
✗ Error: 'nonexistent.txt' does not exist
```

Solution: Your script should check if file exists before attempting deletion.

Error 3: Syntax Error

```
./recycle_bin.sh: line 45: syntax error near unexpected token `fi'
```

Solution: Check for missing `then`, `do`, or unmatched brackets. Use `bash -n script.sh` to find syntax errors.

Error 4: Variable Not Found

```
./recycle_bin.sh: line 67: METADATA_FILE: unbound variable
```

Solution: Ensure all variables are properly initialized. Check spelling.

Error 5: Command Not Found

```
./recycle_bin.sh: line 89: realpath: command not found
```

Solution: Some systems might not have all commands. Provide alternatives or check if command exists.

Error 6: Spaces in Filenames

```
$ ./recycle_bin.sh delete "my file.txt"
# Script breaks or deletes wrong files
```

Solution: Always quote variables: `"$filename"`, `"$path"`, etc.

Error 7: Corrupted Metadata

```
grep: metadata.db: No such file or directory
```

Solution: Always check if files exist and initialize if missing. Implement recovery mechanism.

---

18. Appendix: Advanced Topics (Optional Reading)

18.1 Signal Handling

Trap signals for graceful cleanup:

```
cleanup() {
    echo "Cleaning up..."
    # Cleanup code here
    exit 0
}

trap cleanup SIGINT SIGTERM
```

18.2 Locking Mechanism

Prevent concurrent operations:

```
LOCK_FILE="/tmp/recycle_bin.lock"

acquire_lock() {
    if [ -f "$LOCK_FILE" ]; then
        echo "Another instance is running"
        exit 1
    fi
    touch "$LOCK_FILE"
}

release_lock() {
    rm -f "$LOCK_FILE"
}
```

18.3 Progress Indicators

For long operations:

```
show_progress() {
    local current=$1
    local total=$2
    local percent=$((current * 100 / total))
    echo -ne "Progress: $percent% ($current/$total)\r"
}
```

18.4 Configuration File Parsing

Enhanced config file support:

```
load_config() {
    if [ -f "$CONFIG_FILE" ]; then
        while IFS='=' read -r key value; do
            [[ "$key" =~ ^#.*$ ]] && continue
            [[ -z "$key" ]] && continue
            export "$key=$value"
        done < "$CONFIG_FILE"
    fi
}
```

18.5 Color Output Control

Disable colors for piping:

```
if [ -t 1 ]; then
    # Terminal supports colors
    RED='\033[0;31m'
    GREEN='\033[0;32m'
else
    # Piped or redirected, no colors
    RED=''
    GREEN=''
fi
```

---

19. Appendix: Sample Test Results Template

```
# Recycle Bin System - Test Results

**Student Name:** [Your Name]
**Student ID:** [Your ID]
**Date:** [YYYY-MM-DD]
**Script Version:** 1.0

---

## Test Summary

| Category | Total Tests | Passed | Failed | Pass Rate |
|----------|-------------|--------|--------|-----------|
| Basic Functionality | 13 | 13 | 0 | 100% |
| Edge Cases | 12 | 11 | 1 | 92% |
| Error Handling | 8 | 8 | 0 | 100% |
| Performance | 4 | 4 | 0 | 100% |
| **TOTAL** | **37** | **36** | **1** | **97%** |

---

## Detailed Test Results

### 1. Basic Functionality Tests

#### Test 1.1: Initialize Recycle Bin
- **Status:** ✓ PASS
- **Description:** Verify system initialization creates required directo-
ries
- **Expected:** ~/.recycle_bin/ created with subdirectories
- **Actual:** All directories created successfully
- **Screenshot:** screenshots/init.png

#### Test 1.2: Delete Single File
- **Status:** ✓ PASS
- **Description:** Delete a single file
- **Steps:**
  1. Created test.txt with content
  2. Ran: `./recycle_bin.sh delete test.txt`
  3. Verified file moved to recycle bin
- **Expected:** File moved, metadata created
- **Actual:** Success message displayed, file in recycle bin
- **Screenshot:** screenshots/delete_single.png

[Continue for all tests...]

---

## Known Issues

### Issue 1: Symbolic Link Handling
- **Description:** Symbolic links are followed instead of being moved
- **Impact:** Medium
- **Workaround:** None currently
- **Plan:** Will implement in future version

### Issue 2: Very Long Filenames
- **Description:** Filenames over 255 characters cause truncation in dis-
play
- **Impact:** Low (display only, functionality works)
- **Workaround:** Use ID for operations
- **Plan:** Implement better truncation algorithm

---

## Performance Observations

- Delete operation: ~0.1s per file
- List operation with 100 items: ~0.3s
- Search operation: ~0.2s
- Restore operation: ~0.15s per file

---

## Conclusion

The recycle bin system successfully implements all required core features
with a 97% test pass rate. One edge case (symbolic links) requires future
enhancement. The system performs well under normal operating conditions
and handles errors gracefully.
```

---

20. Appendix: Code Review Checklist

Before submission, review your code against this checklist:

Script Header

- [ ] Shebang line present (`#!/bin/bash`)
- [ ] Author name and date
- [ ] Brief description
- [ ] Version number

Global Variables

- [ ] All uppercase naming
- [ ] Initialized at top of script
- [ ] Descriptive names
- [ ] No magic numbers (use constants)

Functions

- [ ] Each function has header comment
- [ ] Descriptive function names
- [ ] Single responsibility principle
- [ ] Parameters documented
- [ ] Return codes documented
- [ ] Local variables used where appropriate

Error Handling

- [ ] All inputs validated
- [ ] File existence checked before operations
- [ ] Permission checks before file operations
- [ ] Meaningful error messages
- [ ] Graceful failure (no data loss)
- [ ] Return codes used consistently

Code Quality

- [ ] No code duplication
- [ ] Consistent indentation (2 or 4 spaces)
- [ ] Variables quoted properly
- [ ] No unused variables or functions
- [ ] No hardcoded paths (except HOME)
- [ ] Comments explain "why", not "what"

User Experience

- [ ] Clear, helpful messages
- [ ] Confirmation for destructive operations
- [ ] Progress indication for long operations
- [ ] Helpful error messages with suggestions

- [ ] Consistent command syntax

Security

- [ ] No eval or exec of user input
- [ ] Path traversal prevented
- [ ] Temp files created securely
- [ ] File permissions respected
- [ ] No password/sensitive data in code

Testing

- [ ] All functions tested
- [ ] Edge cases tested
- [ ] Error conditions tested
- [ ] Test results documented

Documentation

- [ ] README complete and clear
- [ ] Technical doc explains design
- [ ] All functions documented in code
- [ ] Usage examples provided
- [ ] Known issues documented

---

21. Appendix: Debugging Guide

Enabling Debug Mode

```
# Method 1: Run with debug flag
bash -x ./recycle_bin.sh delete file.txt

# Method 2: Add to script
set -x  # Enable debug output
# your code here
set +x  # Disable debug output

# Method 3: Conditional debugging
DEBUG=1  # Set at top of script

debug_log() {
    if [ "$DEBUG" = "1" ]; then
        echo "[DEBUG] $1" >&2
    fi
}
```

Common Debugging Techniques

1. Check Syntax

```
bash -n recycle_bin.sh
```

2. Add Echo Statements

```
echo "DEBUG: Variable value is: $variable" >&2
```

3. Check Return Codes

```
some_command
echo "Return code: $?"
```

4. Validate Variables

```
echo "RECYCLE_BIN_DIR: $RECYCLE_BIN_DIR"
echo "METADATA_FILE: $METADATA_FILE"
echo "Number of arguments: $#"
echo "All arguments: $@"
```

5. Use ShellCheck

```
shellcheck recycle_bin.sh
```

Debugging Scenarios

Problem: Script doesn't execute

```
# Check if executable
ls -l recycle_bin.sh
# Make executable
chmod +x recycle_bin.sh
```

Problem: Variables not set

```
# Check if sourced or executed
echo "Script PID: $"
echo "BASH_SOURCE: ${BASH_SOURCE[0]}"
```

Problem: Function not found

```
# Check if function is defined
type function_name
declare -f function_name
```

---

22. Final Project Checklist

Before Submission

Code Completion

- [ ] All mandatory features implemented
- [ ] All optional features (if any) implemented
- [ ] No syntax errors (`bash -n`)
- [ ] No shellcheck warnings (critical ones)
- [ ] Script is executable
- [ ] Help command works

Testing

- [ ] All test cases executed
- [ ] Test results documented
- [ ] Screenshots taken
- [ ] Automated test suite runs
- [ ] Known issues documented

Documentation

- [ ] README.md complete
- [ ] TECHNICAL_DOC.md complete
- [ ] TESTING.md complete with results
- [ ] Code comments thorough
- [ ] All functions documented

Organization

- [ ] Correct directory structure
- [ ] All files named correctly
- [ ] No temporary files included
- [ ] Screenshots organized
- [ ] Archive created correctly

Final Review

- [ ] Tested on clean system
- [ ] Verified all links work
- [ ] Spell-checked documents
- [ ] Student name/ID in all docs
- [ ] Version control history (if using git)

---

23. Submission Cover Sheet

Fill out and include this with your submission:

```
═══════════════════════════════════════════════════════════════
              LINUX RECYCLE BIN PROJECT
                  SUBMISSION COVER SHEET
═══════════════════════════════════════════════════════════════

STUDENT INFORMATION
───────────────────────────────────────────────────────────────
Name:             _____________________________________________
Student ID:       _____________________________________________
Email:            _____________________________________________
Course:           _____________________________________________
Section:          _____________________________________________
Instructor:       _____________________________________________
Submission Date:  _____________________________________________

PROJECT INFORMATION
───────────────────────────────────────────────────────────────
Project Title:    Linux Recycle Bin System
Version:          _____________________________________________
Total Files:      _____________________________________________
Archive Size:     _____________________________________________
```

IMPLEMENTATION STATUS

Core Features:
- ☐ Initialize recycle bin
- ☐ Delete files/directories
- ☐ List recycle bin contents
- ☐ Restore files
- ☐ Search functionality
- ☐ Empty recycle bin
- ☐ Help system

Optional Features Implemented:
- ☐ Statistics dashboard
- ☐ Auto-cleanup
- ☐ Quota management
- ☐ File preview
- ☐ Other: ____

TESTING STATUS

Total Test Cases:	____
Tests Passed:	____
Tests Failed:	____
Pass Rate:	____

KNOWN ISSUES

List any known bugs or limitations:

1. ____

2. ____

3. ____

COLLABORATION DECLARATION

I declare that this work is my own and has been completed in accordance with the academic integrity policy. I have properly cited all sources and assistance received.

Resources Used (list websites, books, tools):

1. ____

2. ____

3. ____

AI Assistance (describe what was assisted, if any):

____

____

STUDENT SIGNATURE

```
Signature: ___________________________ Date: ___________________
```

24. Post-Submission: Next Steps

After submitting your project:

1. **Backup Your Work**
   - Keep a copy of your submission
   - Save to cloud storage
   - Keep git repository (if used)
2. **Reflect on Learning**
   - What was most challenging?
   - What would you do differently?
   - What did you learn?
3. **Portfolio Preparation**
   - Clean up code for portfolio
   - Create better README for public viewing
   - Consider publishing on GitHub
4. **Further Enhancements** (Post-Submission)
   - Add GUI using `dialog` or `zenity`
   - Create man page
   - Package as installable script
   - Add network recycle bin support
   - Integration with file managers
5. **Prepare for Demo** (if required)
   - Practice demonstrating features
   - Prepare to explain design decisions
   - Be ready to answer questions
   - Have backup plan if demo fails

25. Additional Resources

Recommended Books

- "Linux Command Line and Shell Scripting Bible" by Richard Blum
- "Classic Shell Scripting" by Arnold Robbins and Nelson H.F. Beebe
- "Advanced Bash-Scripting Guide" by Mendel Cooper (free online)

Online Courses

- Linux Academy - Bash Scripting
- Udemy - Shell Scripting courses

- Coursera - Linux courses

Communities

- Stack Overflow - shell tag
- Reddit - r/bash, r/linux
- Unix & Linux Stack Exchange

Tools

- ShellCheck - syntax and style checker
- Bashate - Bash style checker
- Git - version control
- VS Code with Bash extension

Contact and Support

For questions or clarification:

Instructor: Pedro Azevedo Fernandes (paf@ua.pt); Nuno Lau (nunolau@ua.pt)

Good luck with your project!

Remember: Start early, test often, document thoroughly, and don't hesitate to ask for help when needed.

Document Version: 1.0
Last Updated: October 09th, 2025
Subject to change - check course website for latest version
