**Sistemas Operativos** **Ano lectivo 2025/2026**

# Trabalho Prático 1

# *Linux Recycle Bin Simulation – Announcement of Project Proposal*[1]

### *Course Information*

**Project Title:** Linux Shell-Based Recycle Bin Implementation
**Programming Language:** Bash Shell Scripting
**Groups:** Two students. One student in a very exceptional and justified case.

---

# 1. Project Overview

## 1.1 Objective

Students will develop a complete recycle bin system for Linux environments using shell scripting and modular functions. The system will replicate core Windows Recycle Bin functionality, including file deletion, restoration, permanent deletion, and metadata management.

## 1.2 Learning Outcomes

Upon completion, students will be able to:

- Write modular shell scripts using functions
- Implement file system operations safely
- Handle metadata and logging
- Create user-friendly command-line interfaces
- Apply error handling and validation techniques
- Understand file permissions and ownership in Linux
- Implement data structures using shell arrays and files

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

# 2. Technical Specifications

## 2.1 Core Features (Mandatory)

### *Feature 1: File Deletion (Move to Recycle Bin)*

- **Function Name:** `delete_file()`
- **Description:** Move files/directories to the recycle bin instead of permanent deletion
- **Requirements:**
  - Accept single or multiple file paths as arguments
  - Preserve original file metadata (timestamp, permissions, owner)
  - Store original absolute path for restoration
  - Generate unique identifiers to prevent name conflicts
  - Support both files and directories (recursive)
  - Create timestamped entries in metadata log

### *Feature 2: List Recycle Bin Contents*

- **Function Name:** `list_recycled()`
- **Description:** Display all items currently in the recycle bin
- **Requirements:**
  - Show original filename and path
  - Display deletion date and time
  - Show file size
  - Provide a unique identifier for each item
  - Support formatted output (table view)
  - Implement sorting options (by date, name, size)

### *Feature 3: Restore Files*

- **Function Name:** `restore_file()`
- **Description:** Restore files to their original locations
- **Requirements:**
  - Accept file identifier or pattern matching
  - Restore to the original path
  - Preserve original permissions and timestamps
  - Handle conflicts if the original path is occupied
  - Support selective and batch restoration
  - Provide confirmation prompts for safety

### *Feature 4: Permanent Deletion (Empty Recycle Bin)*

- **Function Name:** `empty_recyclebin()`
- **Description:** Permanently delete items from recycle bin
- **Requirements:**
  - Delete all items permanently

- Support selective deletion by ID or pattern
- Require confirmation before execution
- Update metadata log appropriately
- Provide summary of deleted items

### *Feature 5: Search Functionality*

- **Function Name:** `search_recycled()`
- **Description:** Search for specific items in the recycle bin
- **Requirements:**
  - Search by filename pattern
  - Search by date range
  - Search by file type/extension
  - Display matching results with full details

## 2.2 System Architecture

### *Directory Structure*

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

**Metadata Format**

Each entry must contain:

- Unique ID (timestamp-based or UUID)
- Original filename
- Original absolute path
- Deletion timestamp
- File size
- File type (file/directory)
- Original permissions
- Original owner

## 2.3 Additional Features (Optional - Extra Credit)

1. **Storage Quota Management**
   - Implement size limits for the recycle bin
   - Auto-delete the oldest files when the quota is exceeded
   - Function: `check_quota()`
2. **Auto-cleanup**
   - Automatically delete files older than N days
   - Configurable retention period
   - Function: `auto_cleanup()`
3. **File Preview**
   - Display file content preview for text files

   - Show file information for other types
   - Function: `preview_file()`
4. **Statistics Dashboard**
   - Total files in the recycle bin
   - Total storage used
   - Breakdown by file type
   - Function: `show_statistics()`
5. **Recycle Bin Protection**
   - Password protection for sensitive operations
   - Encryption of metadata
   - Function: `secure_operation()`

---

# 3. Implementation Requirements

## 3.1 Code Structure

### Main Script Organization

```
#!/bin/bash

# Global Variables
RECYCLE_BIN_DIR="$HOME/.recycle_bin"
METADATA_FILE="$RECYCLE_BIN_DIR/metadata.db"

# Function declarations
initialize_recyclebin()
delete_file()
restore_file()
list_recycled()
empty_recyclebin()
search_recycled()
display_help()


# Main program logic

main()
```

## 3.2 Function Requirements

Each function must include:

- Descriptive header comment explaining purpose
- Parameter validation and error checking
- Appropriate return codes (0 for success, non-zero for errors)
- User feedback messages
- Logging of operations

### 3.3 Error Handling

Students must implement robust error handling for:

- Non-existent files
- Permission denied errors
- Disk space issues
- Invalid arguments
- Corrupted metadata
- Path conflicts during restoration

### 3.4 User Interface

The program must support:

- Command-line arguments (e.g., `recycle_bin.sh delete file.txt`)
- Interactive mode with a menu
- Help documentation (`--help` flag)
- Verbose mode for debugging (`--verbose` flag)

---

# 4. Deliverables

### 4.1 Source Code

- Main script file: `recycle_bin.sh`
- All code must be properly commented
- Use meaningful variable and function names
- Follow shell scripting best practices

### 4.2 Documentation

#### *README.md*

- Installation instructions
- Usage examples for all features
- Configuration options
- Troubleshooting guide

#### *Technical Documentation*

- System architecture diagram
- Function flowcharts
- Metadata schema description
- Design decisions explanation

### 4.3 Testing Documentation

- Test cases for each feature
- Test results and screenshots
- Edge cases identified and tested
- Known limitations

### 4.4 Demonstration Video (Optional)

- 5-10 minute walkthrough
- Demonstrate all core features
- Show error handling scenarios

---

# 5. Evaluation Criteria

### Functionality (40%)

- All core features are working correctly (30%)
- Error handling and edge cases (10%)

### Code Quality (25%)

- Proper use of functions and modularity (10%)
- Code readability and comments (8%)
- Following shell scripting conventions (7%)

### Documentation (20%)

- Complete and clear README (10%)
- Technical documentation quality (10%)

### Testing (10%)

- Comprehensive test cases (5%)
- Test coverage and results (5%)

### Innovation (5%)

- Implementation of optional features
- Creative solutions to problems
- User experience enhancements

---

# 6. Submission Guidelines

## 6.1 File Structure

```
StudentName_RecycleBin/
├── recycle_bin.sh
├── README.md
├── TECHNICAL_DOC.md
├── TESTING.md
└── screenshots/
```

## 6.2 Submission Format

- Compressed archive (.tar.gz or .zip)
- Include all documentation and source files
- No binary or temporary files
- Name of the file: SO-2526-T1-Px-Gy-11111-22222.tgz, where:
  - Px is the number of the practical class (P1..P6);
  - Gy is the number of the group within the practical class;
  - 11111 and 22222 are the mec. numbers of the students in the group.

## 6.3 Deadline

**31st October, 2025**

---

# 7. Resources and References

## Shell Scripting Resources

- Advanced Bash-Scripting Guide
- GNU Bash Reference Manual
- Linux file system hierarchy documentation

## Recommended Tools

- `shellcheck` for script validation
- `bash -x` for debugging
- `git` for version control

## System Commands to Study

- `mv`, `rm`, `cp`
- `find`, `stat`, `file`
- `date`, `du`, `df`
- `grep`, `awk`, `sed`

# 8. Safety and Ethics Guidelines

Students must ensure:

- The script never causes data loss
- Proper permissions are respected
- No security vulnerabilities are introduced
- The system doesn't interfere with actual system operations
- Testing is done in safe, isolated environments

---

## *9. Bonus Challenges*

For advanced students:

1. Implement the recycle bin as a system service
2. Create a GUI wrapper using `dialog` or `zenity`
3. Add network recycle bin support for shared directories
4. Implement file versioning (multiple deletions of the same file)
5. Create integration with file managers (Nautilus, Dolphin)

---

## *10. Academic Integrity*

This is a group project. While students from different groups may discuss concepts, all code must be the original work of each group. Use of AI assistants should be limited to debugging and understanding concepts, not generating complete solutions. Proper citation is required for any external code snippets used or any concept from external resources, both from the web or any other sources. The reference methodology will follow the IEEE templates.

**Plagiarism will not be accepted**. Failure to comply with this rule will immediately imply the exclusion of the work.

---