universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Sistemas Operativos **Ano lectivo 2025/2026**

Trabalho Prático 1

Linux Recycle Bin Simulation – Frequently Asked Questions (FAQ)[1]

📋 *Table of Contents*

General Questions

Q1: Can I use Python/Perl/another language instead of Bash?

A: No. This project must be implemented in Bash shell scripting. The learning objectives specifically target shell scripting skills.

Q2: Can I work with a partner?

A: Of course, this is a group project of two students. You may discuss concepts with other group classmates, but all code must be the work of your group only.

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

Q3: How much time should this take?

A: Estimate 15-20 hours total over 3-4 weeks:

- Week 1: 5-6 hours (setup and delete)
- Week 2: 5-6 hours (list and restore)
- Week 3: 5-6 hours (search, empty, testing)
- Week 4: 4-5 hours (documentation and polish)

Q4: What if I can't finish all features?

A: Submit what you have. Partial credit is awarded. A working script with 4 out of 6 features and good documentation can still earn a good grade.

Q5: Can I use AI tools like ChatGPT?

A: You may use AI tools to:

- Understand concepts
- Debug specific errors
- Learn syntax

You may NOT use AI to:

- Generate complete functions
- Write your entire script
- Complete the project for you

Always cite AI assistance in your README.

Q6: Is there a minimum or maximum lines of code?

A: No specific requirement. A complete solution typically ranges from 400-800 lines, including comments. Focus on quality, not quantity.

Q7: Can I add features not in the specification?

A: Yes! Additional features can earn bonus points. Make sure all required features work first.

Technical Questions

Q8: What version of Bash should I use?

A: Bash 4.0 or higher. Check with:

```
bash --version
```

Q9: Will my script work on macOS?

A: macOS uses BSD utilities, which have a different syntax. If developing on macOS:

```
# Linux
stat -c %s file.txt

# macOS
stat -f%z file.txt
```

You may need to detect the OS:

```
if [[ "$OSTYPE" == "darwin"* ]]; then
    # macOS code
else
    # Linux code
fi
```

Q10: Can I use external libraries or packages?

A: Only standard Linux utilities are allowed:

- ✓ mv, rm, cp, stat, date, grep, awk, sed
- ✗ trash-cli, custom packages, Python libraries

Q11: How do I handle files with commas in the name?

A: Options:

1. Use a different delimiter (pipe `|` or tab)
2. Escape commas in filenames
3. Use a more robust format (JSON, but not recommended)

Example with pipe delimiter:

```
echo "$id|$name|$path|$date" >> "$METADATA_FILE"
```

Q12: Should I use absolute or relative paths?

A: Always store absolute paths in metadata. Use `realpath` to convert:

```
abs_path=$(realpath "$file")
```

Q13: What about symbolic links?

A: Decide whether to:

- Follow the link (delete target): Use `-L` flag
- Delete the link itself: Use `-P` flag (default)

Document your choice in technical documentation.

Implementation Questions

Q14: How do I generate a truly unique ID?

A: Use timestamp plus random string:

```
unique_id="$(date +%s%N)_$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 6
| head -n 1)"
```

This gives nanosecond precision plus randomness.

Q15: What if two files have the same name?

A: Use unique IDs for storage, not original filenames. Multiple `document.txt` files can coexist with IDs:

- `1696234567_abc123` (first document.txt)
- `1696234890_def456` (second document.txt)

Q16: How do I preserve file permissions?

A: Store in metadata and restore:

```
# Save permissions
perms=$(stat -c %a "$file")
echo "$unique_id,$filename,$path,$date,$size,file,$perms,$owner" >> "$ME-
TADATA_FILE"

# Restore permissions
chmod "$perms" "$restored_file"
```

Q17: How do I handle very large files?

A: Your script will handle them automatically, but consider:

- Check available disk space first
- Warn user if file is very large
- Optional: Implement compression

Q18: What if the original directory was deleted?

A: During restore:

```
dir_path=$(dirname "$original_path")
if [ ! -d "$dir_path" ]; then
    mkdir -p "$dir_path"
```

```
fi
```

Q19: How do I handle files being deleted twice?

A: If same file is deleted multiple times (after restoration), each deletion gets a unique ID. Your metadata will have multiple entries with same filename but different IDs and dates.

Q20: Should I support wildcards in delete?

A: Yes, your script should support:

```
./recycle_bin.sh delete *.txt
./recycle_bin.sh delete file1.txt file2.txt file3.txt
```

The shell expands wildcards before passing to your script.

Testing Questions

Q21: How many test cases do I need?

A: Minimum 15 test scenarios covering:

- Basic functionality (5-6 tests)
- Edge cases (5-6 tests)
- Error conditions (4-5 tests)

Q22: Do I need an automated test suite?

A: Highly recommended for testing grade. A basic test script can earn extra points:

```
#!/bin/bash
echo "Test 1: Delete file"
echo "test" > test.txt
./recycle_bin.sh delete test.txt
[ ! -f test.txt ] && echo "PASS" || echo "FAIL"
```

Q23: What screenshots should I include?

A: Minimum 5 screenshots:

1. Delete operation with success message
2. List view (normal mode)
3. List view (detailed mode)
4. Restore operation
5. Search results or statistics

Q24: How do I test on a "clean" system?

A: Options:

1. Use a virtual machine
2. Create a new user account
3. Test in a temporary directory
4. Delete `~/.recycle_bin` between tests

```
# Clean test
rm -rf ~/.recycle_bin
./recycle_bin.sh delete test.txt
```

Q25: What edge cases should I test?

A: Critical edge cases:

- Files with spaces in names
- Files with special characters (!@#$%^&*)
- Hidden files (.bashrc)
- Empty files
- Very large files (>100MB)
- Directories with many files
- Symbolic links
- Files without read permissions
- Non-existent files
- Restoring when original path has new file

Documentation Questions

Q26: How detailed should my README be?

A: Include:

- Installation steps (even if simple)
- At least 5 usage examples
- Configuration options
- Known limitations
- Your name and student ID

Aim for 2-3 pages.

Q27: What goes in TECHNICAL_DOC.md vs README.md?

A:

- **README.md:** User guide (how to use)

- **TECHNICAL_DOC.md:** Developer guide (how it works)

README is for users, TECHNICAL_DOC is for developers/graders.

Q28: Do I need to draw diagrams?

A: At least one architecture diagram. ASCII art is fine:

```
┌──────────────────────────────┐
│ User runs command            │
└──────────────────────────────┘
           │
           ▼
┌──────────────────────────────┐
│ main() parses arguments      │
└──────────────────────────────┘
           │
           ▼
┌──────────────────────────────┐
│ Call appropriate function    │
└──────────────────────────────┘
           │
           ▼
┌──────────────────────────────┐
│ Update files/metadata        │
└──────────────────────────────┘
```

Q29: How do I cite sources?

A: In your code:

```
# Source: Stack Overflow - username (URL)
# Adapted from: Advanced Bash Scripting Guide
```

In README, add References section:

```
## References
1. Advanced Bash-Scripting Guide - https://tldp.org/LDP/abs/html/
2. Stack Overflow post about CSV parsing - URL
```

Q30: Can I write documentation after coding?

A: You can, but it's harder. Better to:

- Write README skeleton first (sections, no content)
- Add usage examples as you implement features
- Document design decisions as you make them
- Complete documentation in the final week

Grading Questions

Q31: What's the most important part for grading?

A: Functionality (40 points). A working script with poor documentation scores higher than great documentation with broken code.

Priority:

1. Core features work (40 pts)
2. Error handling (part of 25 pts code quality)
3. Documentation (20 pts)
4. Testing (10 pts)

Q32: How is code quality graded?

A: Graders look for:

- Functions are properly defined and used
- Variables quoted correctly
- Meaningful error messages
- Consistent code style
- Good variable names
- No code duplication

Q33: Will I lose points for not implementing optional features?

A: No. Optional features only give bonus points. You can earn 100/100 without any optional features.

Q34: How much are comments worth?

A: 6 points out of 100. But good comments make grading easier, which helps you get benefit of the doubt on borderline decisions.

Q35: Can I get partial credit?

A: Yes! Grading rubric awards partial points. A partially working delete function can earn 5-7 out of 10 points.

Q36: What if my script works on my computer but not the grader's?

A: Test on a clean Linux system or lab computer before submitting. Common issues:

- Using non-standard commands
- Hard-coded paths specific to your system
- macOS vs Linux command differences

Q37: Is there a penalty for late submission?

A: Check your syllabus. Typical penalty: 10% per day late.

Troubleshooting

Q38: My script says "Permission denied"

A: Two possible issues:

1. Script not executable:

```
chmod +x recycle_bin.sh
```

2. Trying to delete file without permissions:

```
# Check file permissions
ls -l filename

# Your script should detect this
if [ ! -r "$file" ] || [ ! -w "$file" ]; then
    echo "Error: Permission denied"
    return 1
fi
```

Q39: Files with spaces don't work

A: Always quote variables:

```
# WRONG
mv $file $destination

# CORRECT
mv "$file" "$destination"
```

Also in conditions:

```
if [ -f "$file" ]; then
```

Q40: My metadata file gets corrupted

A: Likely issues:

- Filenames contain commas (use a different delimiter)
- Not handling special characters in filenames
- Concurrent access (two instances running)

Solution: Use a better delimiter or escape special characters.

Q41: "Command not found" error

A: Check if the command exists:

```
which realpath
command -v realpath
```

If missing on macOS:

```
# Install coreutils
brew install coreutils

# Or use alternative
readlink -f "$file"  # May not work on all systems
```

Q42: Script works in terminal but not when double-clicked

A: Scripts need to be run from terminal with arguments:

```
cd /path/to/script
./recycle_bin.sh delete file.txt
```

Not designed for double-click execution (needs CLI arguments).

Q43: Getting "bad substitution" error

A: Using bash-specific syntax, but the script runs with sh:

```
# Make sure shebang is correct
#!/bin/bash
# NOT #!/bin/sh
```

Q44: Restore doesn't preserve permissions

A: Two issues:

1. Not storing permissions in metadata
2. chmod syntax wrong

```
# Store
perms=$(stat -c %a "$file")

# Restore
chmod "$perms" "$restored_file"  # Use quotes!
```

Q45: "No such file or directory" when metadata exists

A: Check:

1. Path in metadata is absolute
2. Variable is properly expanded
3. File ID actually exists in files/ directory

```
# Debug
echo "Looking for: $FILES_DIR/$file_id"
ls -la "$FILES_DIR/$file_id"
```

Submission Questions

Q46: What format should my submission be?

A: Compressed archive (.tar.gz or .zip):

```
# Create tar.gz
tar -czf YourName_RecycleBin.tar.gz YourName_RecycleBin/

# Or create zip
zip -r YourName_RecycleBin.zip YourName_RecycleBin/
```

Q47: What should I name my files?

A: Use your actual name:

- `JohnSmith_RecycleBin.tar.gz`
- NOT `recycle_bin.tar.gz` or `project.tar.gz`

Q48: Can I submit a GitHub link instead?

A: Check with your instructor. Usually need to submit files directly, but can include GitHub link in the README as supplementary.

Q49: What if I submit the wrong file?

A: Contact the instructor immediately. Most systems allow resubmission before deadline.

Q50: Should I include the .recycle_bin directory in the submission?

A: NO. Only include:

- Source code (recycle_bin.sh)
- Documentation (*.md files)
- Test suite (test_suite.sh)
- Screenshots

Don't include:

- .recycle_bin/ directory
- Test files
- Temporary files
- .git/ directory (if using git)

Q51: How do I verify my submission is complete?

A: Extract your archive in a new directory and check:

```
# Extract
tar -xzf YourName_RecycleBin.tar.gz
cd YourName_RecycleBin/

# Check structure
ls -la

# Try to run
chmod +x recycle_bin.sh
./recycle_bin.sh help
```

Additional FAQ

Q52: Can I use functions from the starter template?

A: Yes, the starter template is provided to help you. You must implement the logic inside the functions yourself.

Q53: What if I find a bug after submission?

A: Depends on instructor policy. Usually:

- Before deadline: Resubmit
- After deadline: No changes allowed
- Critical bug: Contact the instructor immediately

Q54: How is the demo/presentation graded?

A: If required, typically graded on:

- Ability to demonstrate all features (40%)
- Explanation of design decisions (30%)
- Answering questions about code (30%)

Q55: Can I use a different metadata format (JSON, XML)?

A: CSV is required, but you can use different delimiters. Document your choice clearly.

Q56: Should I handle concurrent operations?

A: Not required. Bonus points if you implement file locking:

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

if [ -f "$LOCK_FILE" ]; then
```

```
    echo "Another instance is running"
    exit 1
fi
touch "$LOCK_FILE"
trap "rm -f $LOCK_FILE" EXIT
```

Q57: What if I disagree with my grade?

A: Follow your institution's grade dispute policy:

1. Review the rubric and your submission
2. Prepare specific questions
3. Schedule a meeting with the instructor
4. Bring evidence of your work

Q58: Can I continue working on this after submission?

A: Absolutely! This makes a great portfolio project:

- Clean up code
- Add more features
- Publish on GitHub
- Include in resume/portfolio

Q59: Will this help me get a job?

A: System programming skills are valuable. This project demonstrates:

- File system operations
- Metadata management
- Error handling
- Documentation skills
- Testing practices

Add to GitHub with a good README for a portfolio.

Q60: Where can I get more help?

A: Resources:

1. Office hours: Pedro Azevedo Fernandes: 10am-11am: 4.1.01; 15pm-16pm: 4.2.25.
2. Course discussion forum: https://elearning.ua.pt/mod/forum/view.php?id=1634485
3. Study groups (concept discussion only)
4. Online resources (cited properly)
5. Man pages (`man bash`, `man stat`, etc.)

Quick Reference

Most Common Issues and Solutions

Problem	Solution
Permission denied	`chmod +x recycle_bin.sh`
Spaces in filenames break script	Quote all variables: `"$var"`
Metadata gets corrupted	Use proper CSV parsing with IFS
Can't find deleted files	Use unique IDs, not filenames
Restore fails	Check if original directory exists
Script runs slow	Don't use loops unnecessarily
Syntax error	Run `bash -n script.sh`
Command not found	Check OS compatibility

Quick Command Reference

```
# Check syntax
bash -n recycle_bin.sh

# Debug mode
bash -x recycle_bin.sh delete file.txt

# Make executable
chmod +x recycle_bin.sh

# Check bash version
bash --version

# Validate with shellcheck
shellcheck recycle_bin.sh

# Create submission
tar -czf Name_RecycleBin.tar.gz Name_RecycleBin/

# Test extraction
tar -tzf Name_RecycleBin.tar.gz
```

Emergency Checklist (Last Day)

- [ ] Script is executable
- [ ] Help command works
- [ ] Delete works
- [ ] List works
- [ ] Restore works
- [ ] README exists with your name
- [ ] At least 2-3 screenshots
- [ ] Submission file named correctly
- [ ] Tested extraction of submission

If 7+ checked, SUBMIT IT!

Contact for Questions

Before emailing, check:

1. This FAQ
2. Project proposal document
3. Quick reference guide
4. Course discussion forum

If still stuck, email with:

- Subject: "Recycle Bin Project - [Specific Issue]"
- Your name and student ID
- Specific question
- What you've tried
- Relevant error messages
- Code snippet (5-10 lines)

Do NOT send:

- "My code doesn't work, please fix"
- Entire script
- Screenshots of code (send text)
- Questions answered in documentation

Final Thoughts

Remember:

- 💡 Start early, ask questions
- 🐛 Test frequently, debug systematically
- 📝 Document as you go
- 💾 Save backups
- 🎯 Core features first, extras later
- ✅ Something working is better than nothing
- 🤝 Concepts with others, code alone
- 🎓 Learn from mistakes

This Project Teaches:

- Practical shell scripting
- File system operations
- Metadata management
- Error handling
- Professional documentation

- Testing methodologies
- Real-world programming

Beyond This Course:

These skills apply to:

- System administration
- DevOps automation
- Build scripts
- Deployment scripts
- Data processing pipelines

You've got this! Good luck! 🚀

FAQ Last Updated: 09th October 2025 – Check eLearning website for updates

---