**Technical**

UNIX

Project: Recycle and Restore

© FDM Group Ltd 2019. All Rights Reserved.

Any unauthorized reproduction or distribution in part  
or in whole will constitute an infringement of copyright.

**Problem:** UNIX has no recycle bin at the command line. When you remove a file or directory, it is gone and cannot be restored. This project is to write a recycle script and a restore script. This will provide users with a recycle bin which can be used to safely delete and restore files.

## Phase 1 – Basic Functionality

Write a script called recycle that mimics the rm command. The recycle script should be able to accept the name of a file as a command line argument as rm does, but instead of deleting the file, your script should move it to a recyclebin directory called recyclebin in your home directory.

1. **The script name is recycle and will be stored in $HOME/project.**
2. The recycle bin will be **$HOME/recyclebin**. Ensure that *your script* creates this directory.
3. The file to be deleted will be a command line argument and the script should be executed as follows: **bash recycle fileName**
4. The script must test for the following error conditions and **display the same error messages as the rm command**.

* No filename provided - Display an error message if no filename is provided as an argument, and set an error exit status.
* File does not exist - Display an error message if file supplied does not exist, and terminate the script.
* Directory name provided - Display an error message if a directory name is provided instead of a filename, and terminate the script.
* Test that the file being deleted is not the recycle script. If it is, display the error message “Attempting to delete recycle – operation aborted” and terminate the script. Be sure to create a hard link for your script before testing that this works.

1. The filenames in the recyclebin, will be in the following format:

**fileName\_inode**

For example, if a file named f1 with inode 1234 is recycled, the file in the recyclebin will be named f1\_1234. This gets around the potential problem of deleting two files with the same name. The recyclebin will only contain files, not directories.

1. Create a hidden file called **.restore.info in $HOME**. Each line of this file will contain the name of the file in the recyclebin, followed by a colon, followed by the original absolute path of the file. For example, if a file called f1, with an inode of 1234 was recycled from the /home/trainee1 directory, .restore.info will contain:

f1\_1234:/home/trainee1/f1

If another file named f1, with an inode of 5432, was recycled from the /home/trainee1/testing directory, then .restore.info will contain:

f1\_1234:/home/trainee1/f1

f1\_5432:/home/trainee1/testing/f1

## Phase 2 – Basic Restore

Write a script called restore to restore individual files back to their original location.

The user will determine which file is to be restored and use the file name with inode number in order to restore the file. For example: **bash restore f1\_1234**

1. **Script name is restore and stored in $HOME/project**
2. The file to be restored will be a command line argument and the script should be executed as follows:

**bash restore f1\_1234**

This is the name of the file in $HOME/recyclebin.

1. The file must be restored to its original location, using the pathname stored in the .restore.info file.
2. The script should test for the following error conditions and **display the same error messages as the rm command**.

* No filename provided - Display an error message if no file provided as argument, and set an error exit status.
* File does not exist - Display an error message if file supplied does not exist, and terminate the script.

1. The script must check whether the file being restored already exists in the target directory. If it does, the user will be asked “Do you want to overwrite? y/n ”

The script must restore the file if the user types y, Y, or any word beginning with y or Y to this prompt, and not restore it if they type *anything* else.

1. After the file has been successfully restored, the entry in the .restore.info file will be deleted.

## Phase 3 – Multiple Files, Wildcards and Option Flags

The rm command can remove multiple files, for example rm file1 file2 file3. rm can also use wildcards, for example rm file\* . The rm command can use the –i option, for interactive mode, and the –v option for verbose mode. Add this functionality to your recycle script.

1. Ensure the script can recycle multiple files, even if some of the files provided do not exist. Wildcards should work as well. This is how rm works.
2. Update the script to allow the –i option. If used, prompt the user, asking for confirmation, in the same way as rm –i. A response beginning with y or Y means yes. All other responses mean no.
3. Update the script to allow the –v option. Display a message confirming deletion, in the same way as rm –v.
4. Ensure the script works with both options in either order, -iv and -vi.
5. If an invalid option is passed into the script, display an error message which shows the offending option value, and terminate the script with a non-zero exit status, just like the rm command.

## Phase 4 – Recycle Files Recursively

The rm command can remove directories and their contents using the –r option.

Add this functionality to your recycle script.

1. Update the recycle script to support the use of the –r option to remove a directory and recycle its contents.
2. The $HOME/recyclebin directory will have a flat structure, containing only files.

Do not move directories into $HOME/recyclebin. It is only necessary to move files. Once all files have been moved, delete the directory in question. The entry in .restore.info, will record the same file location information as in Phase 1, item 6, for later restoration.

## Phase 5 – Restore Files Recycled Recursively

Update the restore script to enable the restoration of files that were recycled recursively.

1. Ensure your script is able to restore files previously recycled using recycle –r.
2. The script will identify the directory to restore the file to, by accessing the .restore.info file. It will then recreate this directory, if necessary, before restoring the file. Finally, it will remove the appropriate entry in .restore.info.

## Testing

It is essential that you test it as you go along. Be sure you test each piece of functionality. It is important to get each phase working and thoroughly tested before moving onto the next phase. Later phases are dependent on a fully functional phase 1.

When testing, **you need to be aware that the script will be marked by someone logged in as a different user. It is important you remove any references to your personal environment, as they will cause the script to fail when being marked.**

## Coding Standards

The coding standards the script must conform to are:

* Use comments to document the script
* Write tidy code using blank lines and indentation
* Remove all troubleshooting code and commented out code
* **You are not allowed to use the commands ‘sed’ or ‘awk’**

## Backups

**Ensure you backup your scripts. It is highly recommended you create hard links for recycle and restore. Commands for hard links:**

**ln ~/project/recycle ~/recyclebackup**

**ln ~/project/restore ~/restorebackup**

**ln ~/.restore.info ~/.restore.info.backup**

### Marking

Your scripts will be executed to check that they work. **Your trainer will award you 0 points if your project scripts do not execute. Any script whose logic cannot be explained or defended will similarly be awarded 0 points.**

Although you may use course materials and internet resources, you cannot discuss concepts with fellow classmates, share answers, or steal code. Any evidence of collaboration or copying will have serious consequences. Similarly, you cannot post these project specifications anywhere online. Failure to conform to these standards will be a breach of contract.