Anglia Ruskin University

JCL Report

MOD003218- Operating Systems

2370967

1. Table of Content

2. Introduction

Job Control Language (JCL) is a vital tool in automating and managing batch job processing in mainframe systems. It allows users to define and manage the execution of tasks, including input and output management and job sequencing, which is essential for maintaining efficiency in large-scale computing environments. JCL is frequently used in industries such as banking, telecommunications, and enterprise resource planning (ERP) systems, where accurate and reliable data processing is crucial. This assignment explores the design and implementation of two scripts, safe_edit.bat for MS-DOS and safeedit.sh for Linux, aimed at providing a safe and efficient method for editing text files through automatic backup and logging mechanisms. Both scripts are intended to safeguard data integrity by creating backups before editing and ensuring that changes to files are logged, thus minimizing the risk of data loss due to accidental errors.

The MS-DOS batch script caters to users in Windows environments, leveraging common tools like Notepad for text editing, while the Linux Bash script targets users in Linux-based systems, where terminal-based editors such as vi are commonly used. Both scripts support command-line and interactive modes, allowing flexibility in usage. They automate key tasks such as file backup, error handling, and the maintenance of logs to ensure smooth operations. In real-life scenarios, these automated backup systems are highly beneficial for software developers and system administrators who require reliable ways to manage configuration files and avoid human error during manual editing (Heath, 2015). Additionally, such scripts are commonly used in educational settings, where Linux-based systems are prevalent, and students are often tasked with file management in a controlled environment (Tanenbaum, 2016). These use cases highlight the importance of automating repetitive tasks to maintain data consistency and reduce the likelihood of operational failures.

3. Program Design and Implementation

3.1 MS-DOS Batch Script Objectives

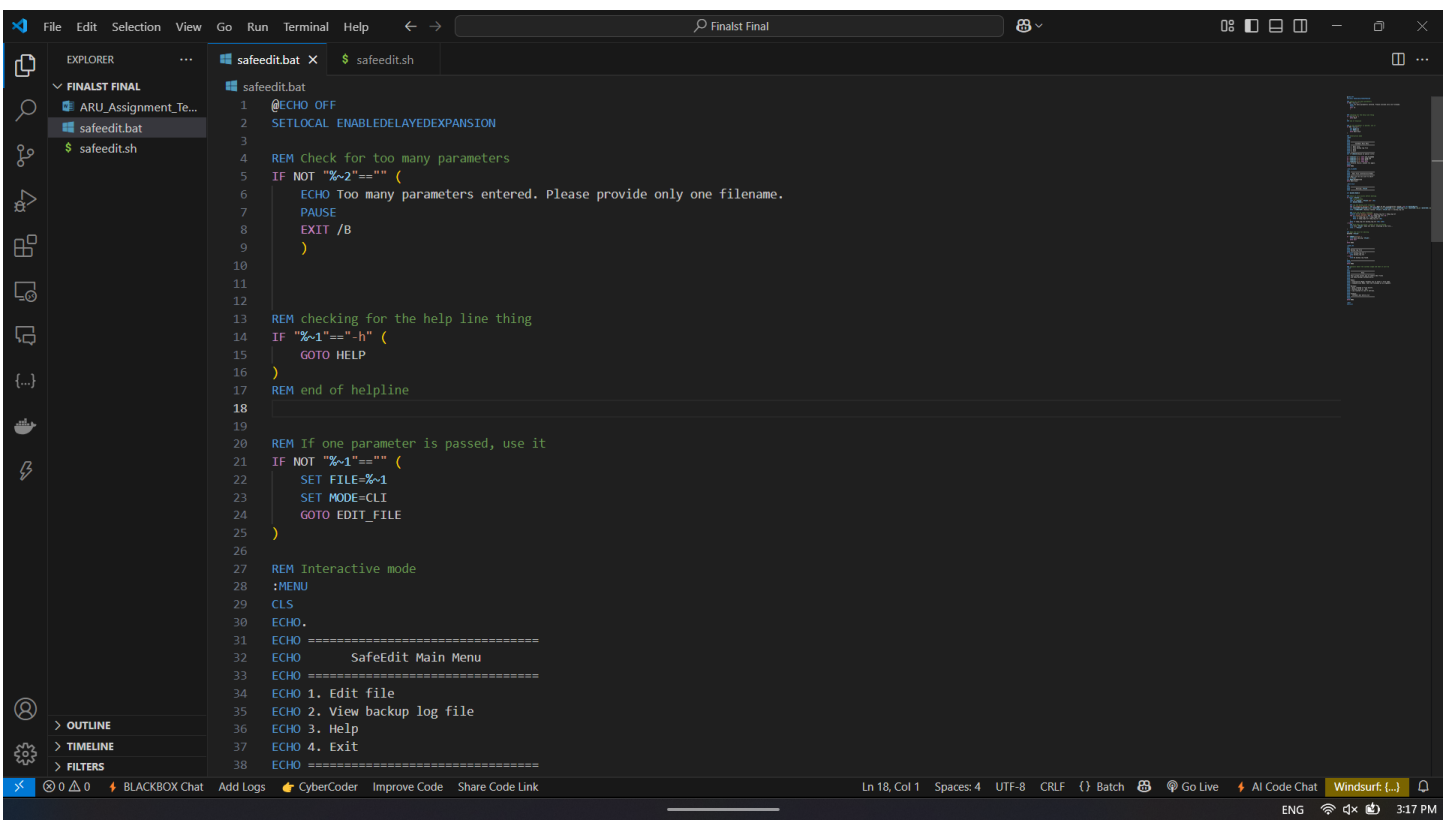
The safe_edit.bat script is designed to provide a safe and user-friendly way to edit text files in a Windows MS-DOS environment. It helps users avoid unintentional data loss by making automatic backups before editing and recording recent changes. The script can be used in two modes.

- **Command-Line Mode** – The user provides a filename as an argument while running the script.
- **Interactive Mode** – A menu interface allows users to choose to edit a file, view backup logs, read help, or exit.

Key Features

- Automatically creates a backup (filename.bak) if the file exists.
- Logs each backup with a timestamp in a file named backup_log.txt.
- Limits the log file to the five most recent backup entries.
- Provides a help section for user guidance.
- Opens files in **Notepad** for editing, a default text editor on Windows systems.

This script is suitable for environments where users work with configuration files or plain text data and needs a basic yet reliable way to preserve changes. Using notepad for editing ensures compatibility with all Windows systems, as it has been the default editor in every version of Windows since the 1980s (Microsoft Docs, 2023). But for me I used visual studio code to edit my MS-DOS document which was a much easier and much visually appealing way for me.



(A screenshot of my MS-DOS script on VSC)

3.2 Linux Bash Script Objectives

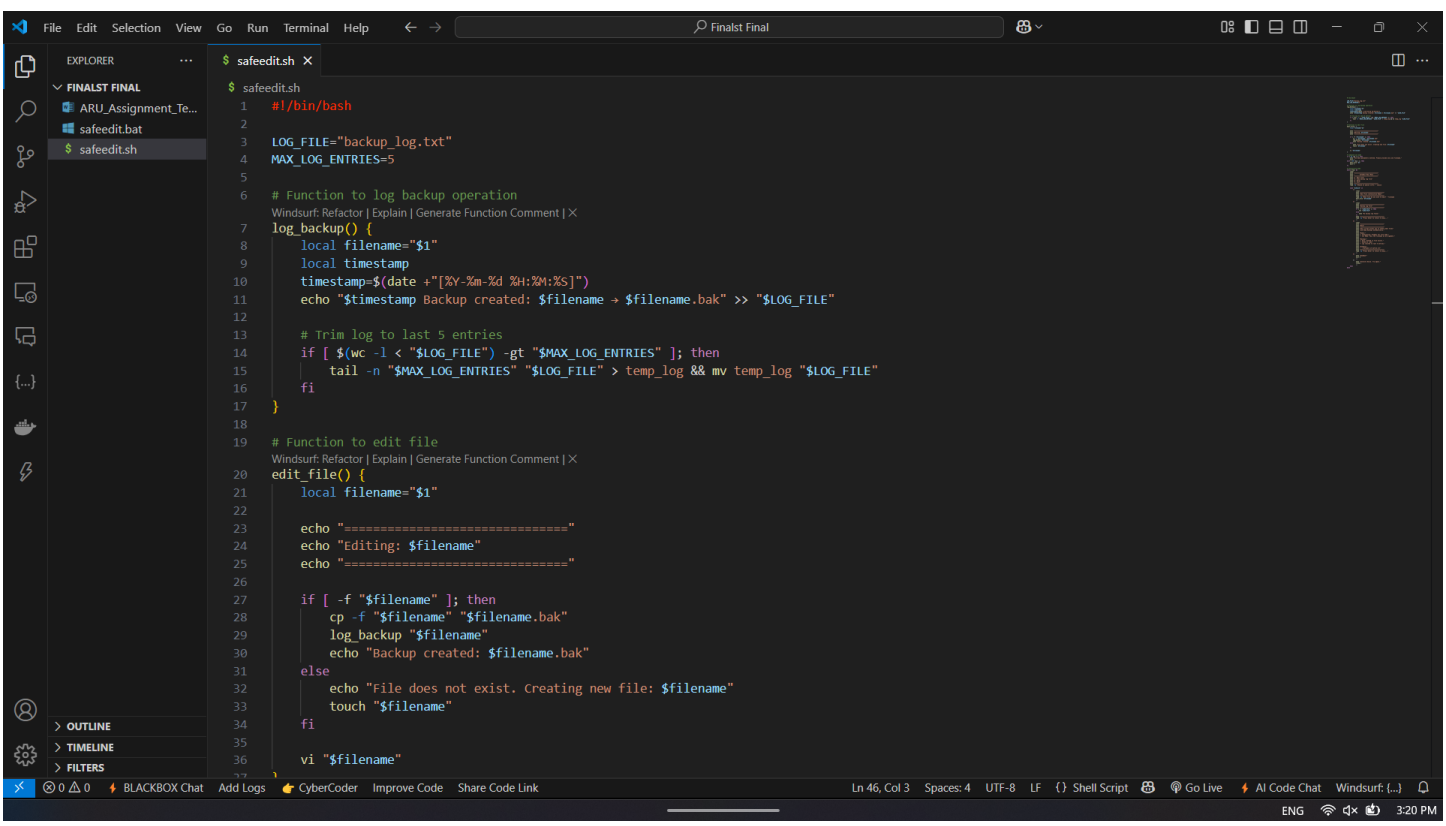
The safeedit.sh script provides a simple and safe method for editing text files on Linux systems. It is especially useful for students or administrators who need to preserve the original state of important files before making manual edits. Like its Windows counterpart, this script supports two modes.

- **Command-Line Mode** – The user provides the filename directly as a script argument.
- **Interactive Mode** – A menu is displayed, allowing the user to select actions like editing a file, viewing backup logs, accessing help, or exiting.

Key Features

- Uses cp to create a backup (filename.bak) before editing an existing file.
- Logs each backup operation with a date and time stamp into backup_log.txt.
- Limits the log to the last five entries for clarity.
- Opens files in the vi editor, aligning with common Linux terminal practices.
- Offers clear prompts and a basic menu system for navigation.

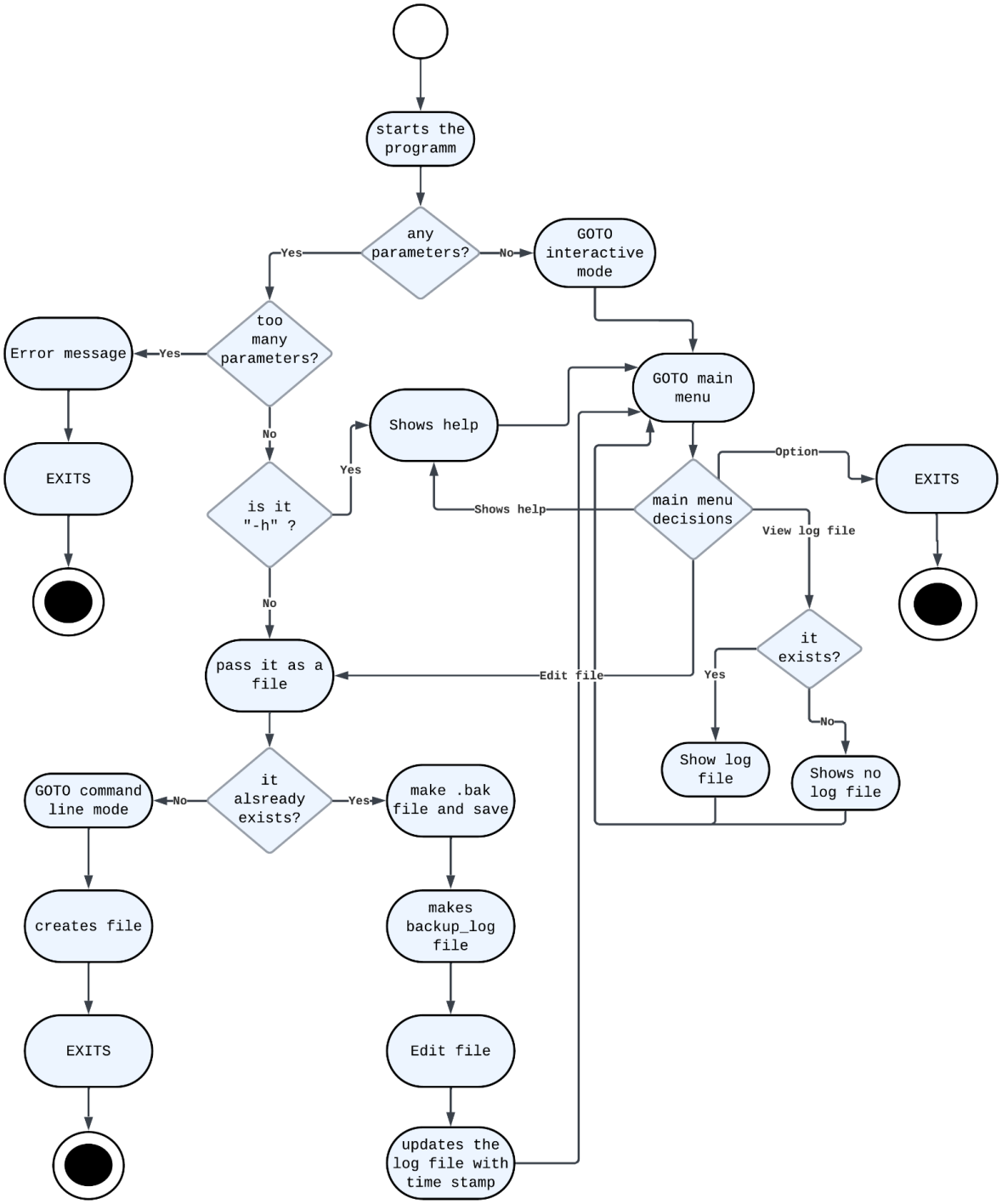
This script is particularly relevant in educational Linux environments, where the use of terminal-based editors like vi is often required in coursework or system administration tasks. According to Red Hat's official documentation, vi remains one of the most essential tools in any Linux distribution, especially for managing text files in a terminal-only interface (Red Hat, 2022). But still, just like for my MS-DOS document for the first editing of the Linux document I used VSC.



(A screenshot of my Linux script on VSC)

3.3 System Design and Flow

The system design for both the MS-DOS and Linux scripts follows a well-defined structure that ensures efficiency and user-friendliness. The program starts by checking for user input, offering both command-line and interactive modes. Key operations include backing up files before editing and logging each backup with a timestamp, ensuring data integrity and traceability. This design facilitates easy navigation through user choices while minimizing the risk of data loss by maintaining up-to-date backups (Sommerville, 2011). The approach prioritizes both reliability and simplicity, making it suitable for environments where file safety is critical.



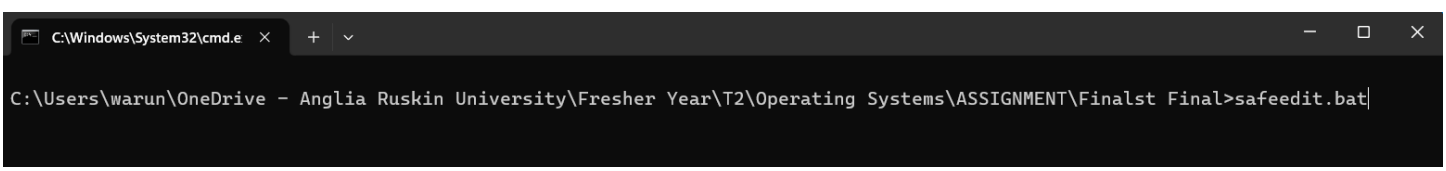
4. Testing and results

4.1 Functionality & Mode Detection

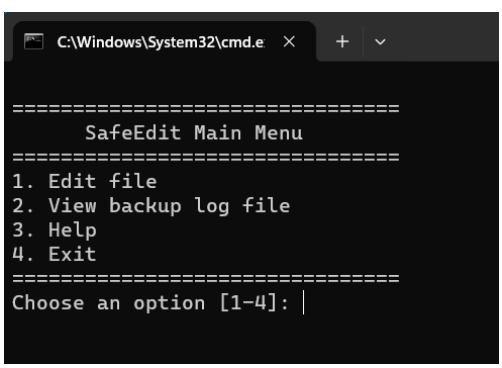
1. **Run without parameters**
   → Expected: Goes to interactive menu mode.

```
-----------------------------------MSDOS CODE SNIPPET-----------------------------------
REM Interactive mode
:MENU
CLS
ECHO.
ECHO ================================
ECHO       SafeEdit Main Menu
ECHO ================================
ECHO 1. Edit file
ECHO 2. View backup log file
ECHO 3. Help
ECHO 4. Exit
ECHO ================================
SET /P CHOICE=Choose an option [1-4]:

IF "%CHOICE%"=="1" GOTO ASK_FILENAME
IF "%CHOICE%"=="2" GOTO VIEW_LOG
IF "%CHOICE%"=="3" GOTO HELP
IF "%CHOICE%"=="4" GOTO EXIT
ECHO Invalid choice, please try again.
PAUSE
GOTO MENU
-----------------------------------MSDOS CODE SNIPPET-----------------------------------
```



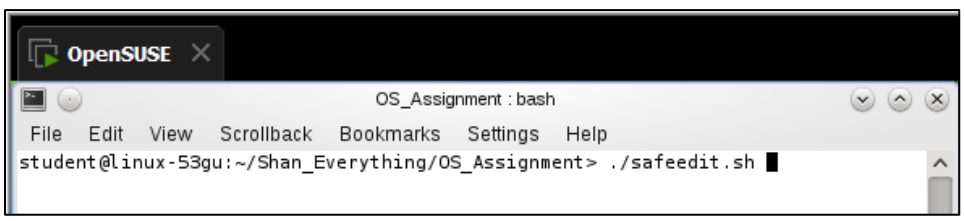
MSDOS Capture 1- Running MSDOS without a parameter



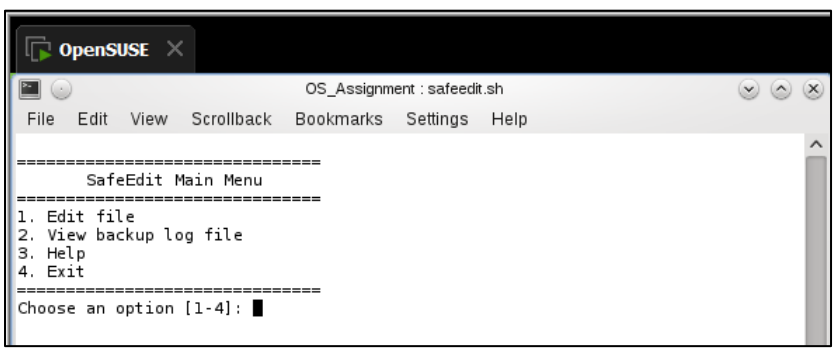
MSDOS Capture 2- Interactive mode

When the user runs the file without a parameter the script will bring the user to the interactive mode. Where the user will meet with the main menu.

```
-------------------------------------BASH CODE SNIPPET-------------------------------------
while true; do
    clear
    echo "==============================="
    echo "       SafeEdit Main Menu      "
    echo "==============================="
    echo "1. Edit file"
    echo "2. View backup log file"
    echo "3. Help"
    echo "4. Exit"
    echo "==============================="
    read -rp "Choose an option [1-4]: " choice
-------------------------------------BASH CODE SNIPPET-------------------------------------
```



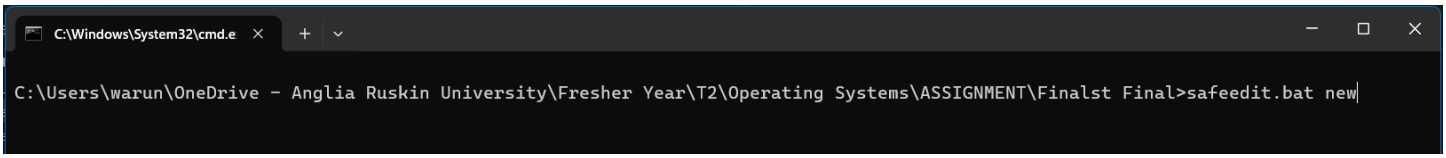
BASH Capture 1- Running bash without a parameter



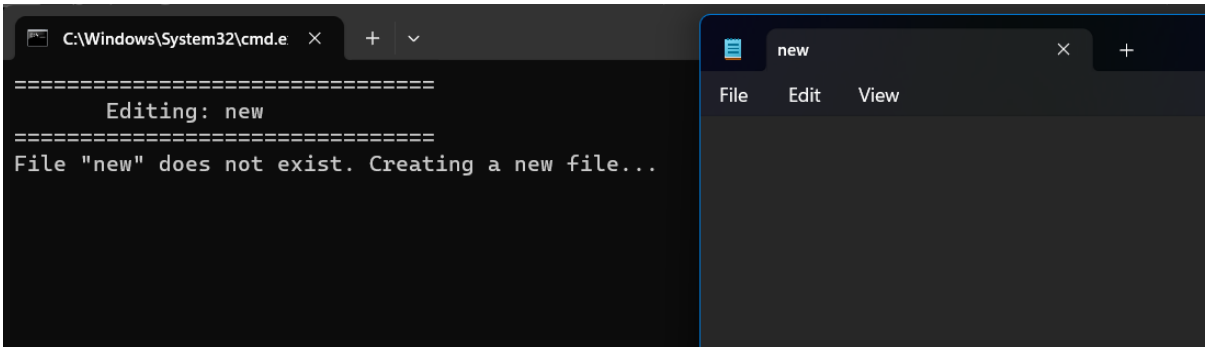
BASH Capture 2- Interactive mode for Bash

2. **Run with one valid filename as parameter**
   → Expected: Goes to command-line mode and opens file for editing.

```
-------------------------------------MSDOS CODE SNIPPET-------------------------------------
REM If one parameter is passed, use it
IF NOT "%~1"=="" (
    SET FILE=%~1
    SET MODE=CLI
    GOTO EDIT_FILE
)
-------------------------------------MSDOS CODE SNIPPET-------------------------------------
```



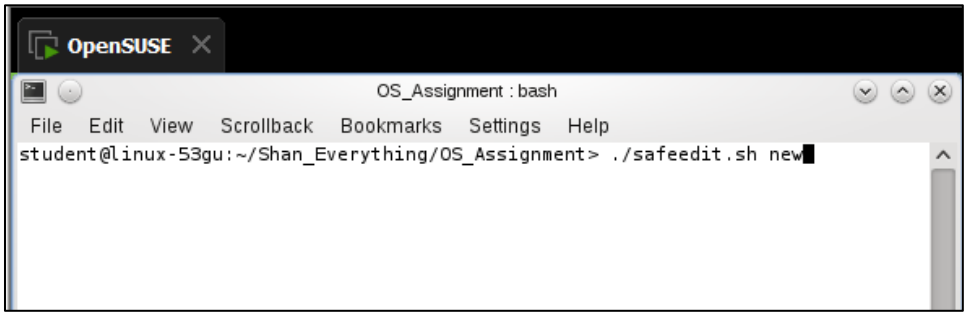
MSDOS Capture 3- giving file name as a parameter



MSDOS Capture 4- script opening the given parameter file in notepad

When the system was given a fie name as a parameter it would check if the file exists. If it does it will open that file. If not, it will make a new file that will open in notepad (or VI editor).

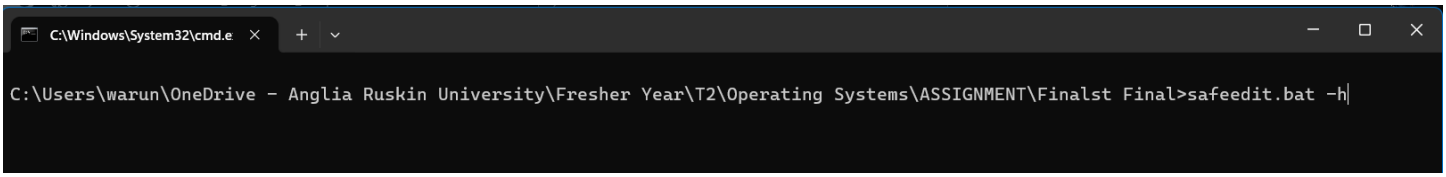
```
-------------------------------------BASH CODE SNIPPET-------------------------------------
elif [ $# -eq 1 ]; then
    edit_file "$1"
    exit 0
fi
-------------------------------------BASH CODE SNIPPET-------------------------------------
```



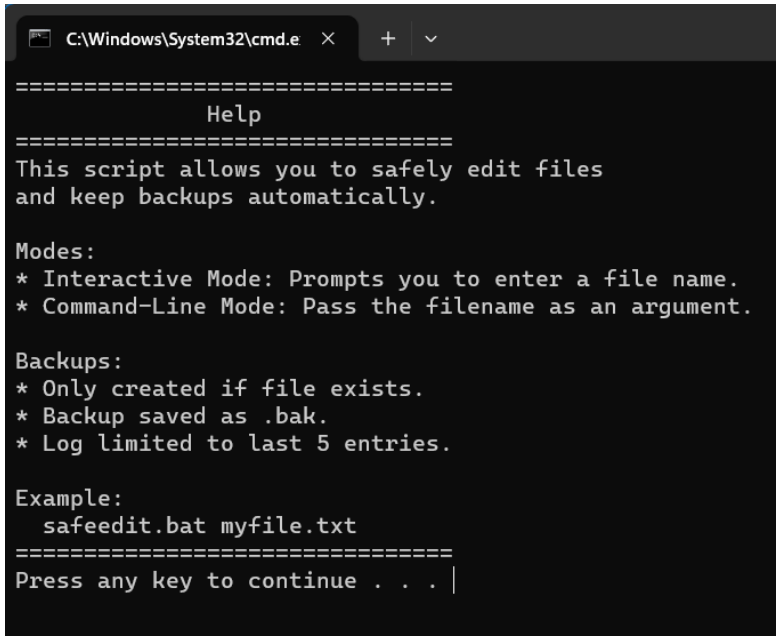
BASH Capture 3- giving the "new" file to the document

3. **Run with -h parameter**
   → Expected: Displays help message and exits.

```
-----------------------------------MSDOS CODE SNIPPET-----------------------------------
REM checking for the help line thing
IF "%~1"=="-h" (
    GOTO HELP
)
-----------------------------------MSDOS CODE SNIPPET-----------------------------------
```



MSDOS Capture 5- -h as the parameter

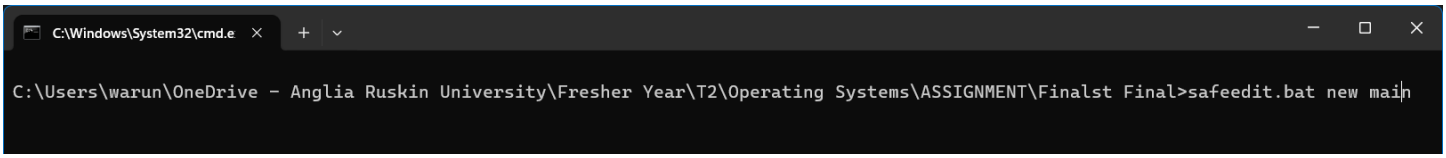


MSDOS Capture 6- display help

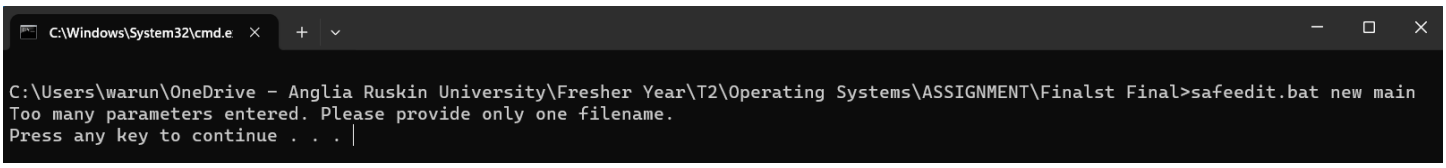
If the scripts gets -h as the first parameter it will show the help from the document. But because ofo the nature of how unix works I was not able to use -h as a parameter and give the help from menu in unix. But the unix script still got the help in its main menu.

4. **Run with more than one parameter**
   → Expected: Error message about too many parameters and exits.

```
-------------------------------------MSDOS CODE SNIPPET-------------------------------------
REM Check for too many parameters
IF NOT "%~2"==""  (
    ECHO Too many parameters entered. Please provide only one filename.
    PAUSE
    EXIT /B
    )
-------------------------------------MSDOS CODE SNIPPET-------------------------------------
```



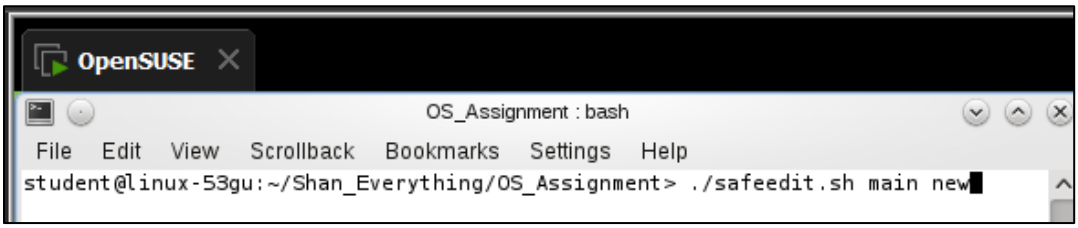
MSDOS Capture 7- Giving more than one parameter



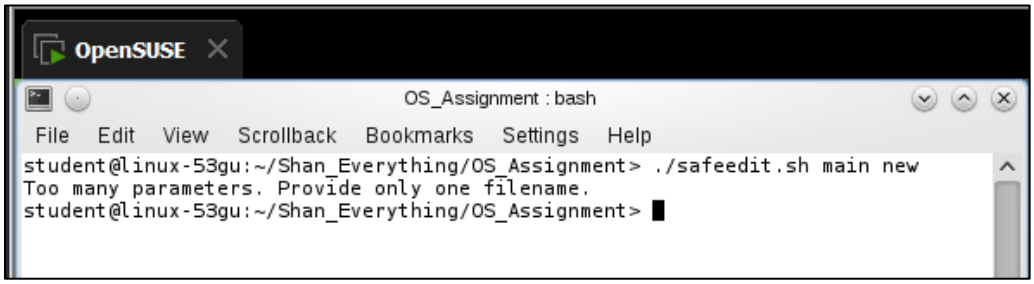
MSDOS Capture 8- Shows an error

Both scripts validate parameter count and exit gracefully if more than one parameter is detected.

```
-------------------------------------BASH CODE SNIPPET-------------------------------------
if [ $# -gt 1 ]; then
    echo "Too many parameters. Provide only one filename."
    exit 1
-------------------------------------BASH CODE SNIPPET-------------------------------------
```



BASH Capture 4- Giving more than one parameter



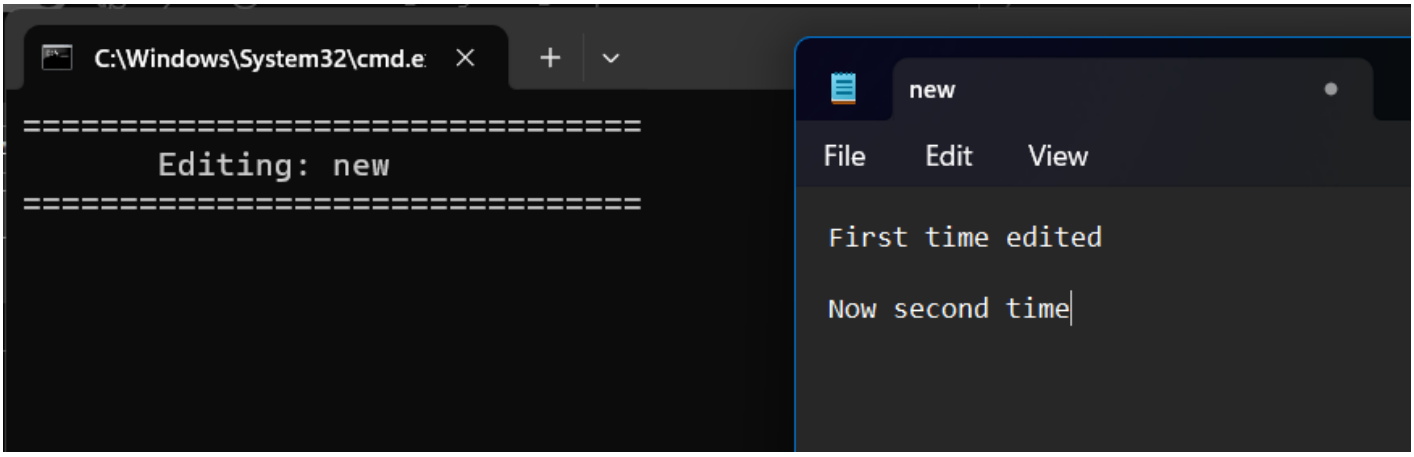
BASH Capture 5- Shows an error

4.2 Backup Handling

5. **Edit existing file (CLI mode)**
   → Expected: Creates .bak backup file and logs it with timestamp.

```
------------------------------------MSDOS CODE SNIPPET------------------------------------
REM Check if file exists before editing
IF EXIST "%FILE%" (
    REM Create backup
    COPY /Y "%FILE%" "%FILE%.bak" >NUL
    SET BACKUP_MADE=1

    REM Log the backup with timestamp
    FOR /F "tokens=2 delims==" %%I IN ('WMIC OS GET LocalDateTime /VALUE') DO SET
DATETIME=%%I
    SET TIMESTAMP=!DATETIME:~0,4!-!DATETIME:~4,2!-!DATETIME:~6,2!
!DATETIME:~8,2!:!DATETIME:~10,2!:!DATETIME:~12,2!
    ECHO [!TIMESTAMP!] Backup created: %FILE% → %FILE%.bak >> backup_log.txt
------------------------------------MSDOS CODE SNIPPET------------------------------------
```



MSDOS Capture 9 – Opens the new file in notepad

```
C:\Windows\System32\cmd.e

C:\Users\warun\OneDrive - Anglia Ruskin University\Fresher Year\T2\Operating Systems\ASSIGNMENT\Finalst Final>dir
 Volume in drive C has no label.
 Volume Serial Number is DA4E-BF2D

 Directory of C:\Users\warun\OneDrive - Anglia Ruskin University\Fresher Year\T2\Operating Systems\ASSIGNMENT\Finalst Final

04/08/2025  08:21 PM    <DIR>          .
04/07/2025  02:52 PM    <DIR>          ..
04/08/2025  08:21 PM           955,444 ARU_Assignment_Template.docx
04/08/2025  08:20 PM                56 backup_log.txt
04/08/2025  08:21 PM                39 new
04/08/2025  08:20 PM                 3 new.bak
04/08/2025  02:26 PM             3,332 safeedit.bat
04/08/2025  08:04 PM             2,713 safeedit.sh
               6 File(s)        961,587 bytes
               2 Dir(s)  133,201,473,536 bytes free

C:\Users\warun\OneDrive - Anglia Ruskin University\Fresher Year\T2\Operating Systems\ASSIGNMENT\Finalst Final>
```

MSDOS Capture 10- Also creates the new.bak file and the backup_log.txt file

```
backup_log.txt
backup_log.txt > [2025-04-08 20:20:21] Backup created: new → new.bak
1   [2025-04-08 20:20:21] Backup created: new → new.bak
2
```

MSDOS Capture 11- Backup_log.txt file would have the time stamp and the details of what backups were made.

The system ensures every edit session preserves the previous version by creating a backup and recording the event in the log. To retrieve the current date and time for logging, the script uses the “WMIC OS GET LocalDateTime /VALUE” command. This approach was somewhat challenging to implement, so I referred to Microsoft’s documentation on WMIC to correctly extract and format the timestamp (Microsoft, 2025).

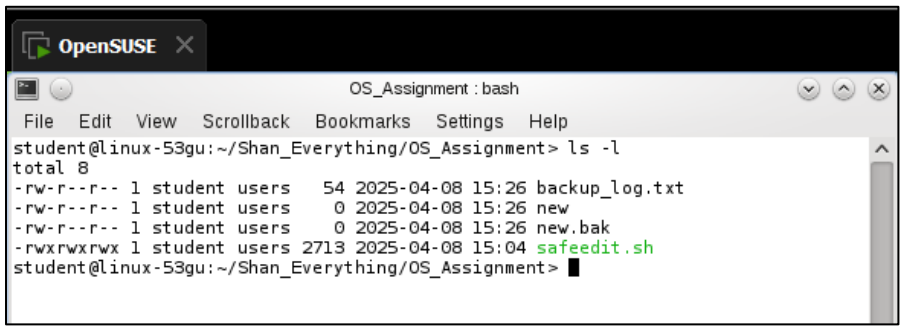
```bash
-------------------------------------BASH CODE SNIPPET-------------------------------------
edit_file() {
    local filename="$1"
    echo "==============================="
    echo "Editing: $filename"
    echo "==============================="

    FILE_EXISTED=0
    if [ -f "$filename" ]; then
        FILE_EXISTED=1
        cp -f "$filename" "$filename.bak"
    else
        touch "$filename"
        echo "File does not exist. Creating new file: $filename"
```

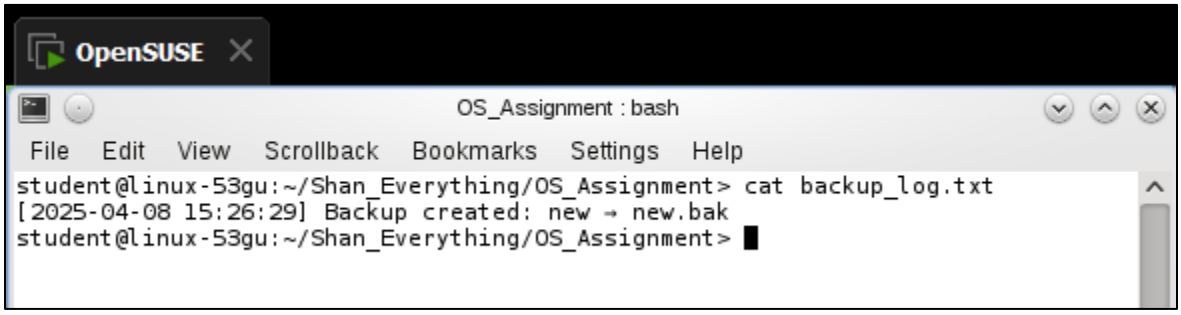
```
    fi

    vi "$filename"

    # After editing, if the file existed BEFORE, log the backup
    if [ "$FILE_EXISTED" -eq 1 ]; then
        log_backup "$filename"
    fi
}
-------------------------------------BASH CODE SNIPPET-------------------------------------
```



BASH 6- it has made the backup_log.txt and the backup file



BASH Capture 7- Backup_log.txt file would have the time stamp and the details of what backups were made.

4.3 Log Management

6. Exceed 5 log entries

→ Expected: Log trims to show only latest 5 entries.

```
-----------------------------------MSDOS CODE SNIPPET-----------------------------------
REM Trim log to last 5 entries
    FINDSTR /R /C:"Backup created:" backup_log.txt > temp_log.txt
    FOR /F "skip=5 delims=" %%L IN (temp_log.txt) DO (
        MORE +1 temp_log.txt > temp_log2.txt
        MOVE /Y temp_log2.txt temp_log.txt >NUL
    )
    MOVE /Y temp_log.txt backup_log.txt >NUL 2>NUL
-----------------------------------MSDOS CODE SNIPPET-----------------------------------
```

```
backup_log.txt ×
backup_log.txt > [2025-04-08 21:18:06] Backup created: new → new.bak
1  [2025-04-08 21:18:06] Backup created: new → new.bak
2  [2025-04-08 21:18:11] Backup created: new → new.bak
3  [2025-04-08 21:18:20] Backup created: new → new.bak
4  [2025-04-08 21:18:25] Backup created: new → new.bak
5  [2025-04-08 21:18:30] Backup created: new → new.bak
6
```

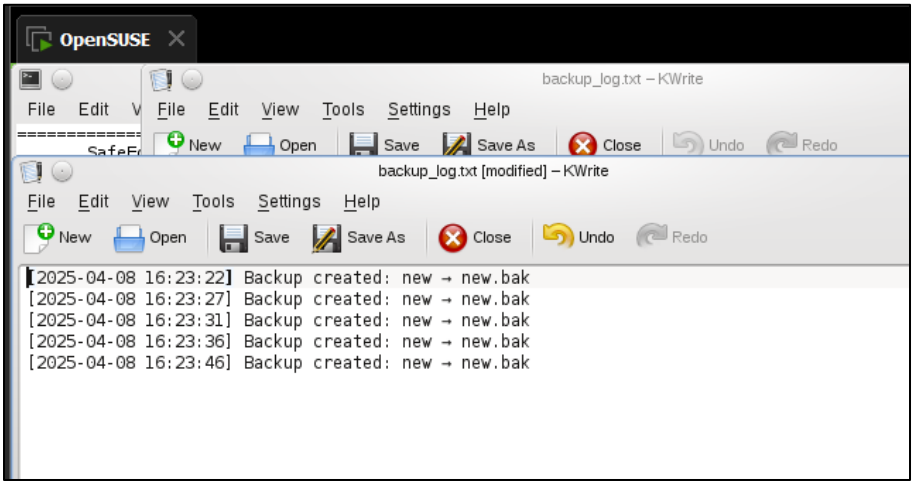
MSDOS Capture12- backup_log.txt saves five log entries.

```
backup_log.txt ×
backup_log.txt > ...
1  [2025-04-08 21:18:11] Backup created: new → new.bak
2  [2025-04-08 21:18:20] Backup created: new → new.bak
3  [2025-04-08 21:18:25] Backup created: new → new.bak
4  [2025-04-08 21:18:30] Backup created: new → new.bak
5  [2025-04-08 21:19:10] Backup created: new → new.bak
6
```

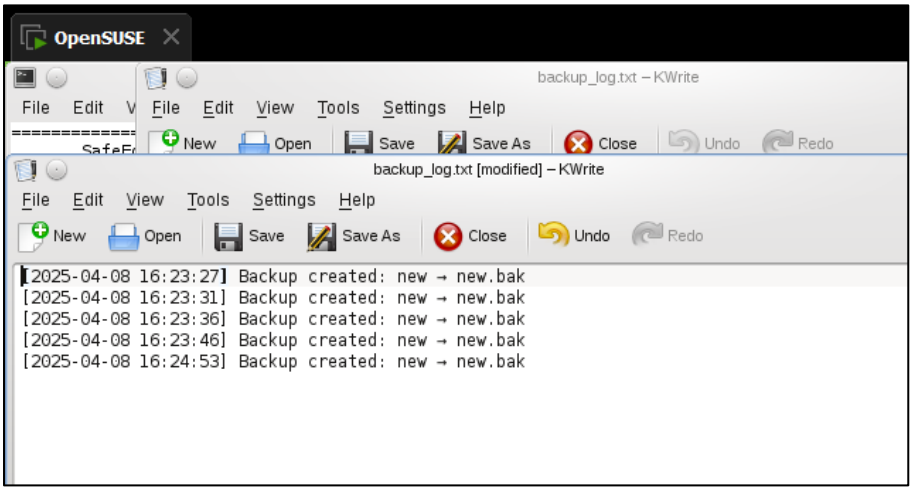
MSDOS Capture 13- When it's the 6th one it deletes the oldest (1st) log entry and adds the 6th log as the 5th log.

Both scripts manage log size by trimming older entries, keeping the log concise and recent. In the MS-DOS script, this process was more complex due to batch scripting limitations. The solution was inspired by a community answer on Stack Overflow that explains how to delete the first few lines of a text file using loops and temporary files (Stack Overflow, 2019).

```
-----------------------------------BASH CODE SNIPPET-----------------------------------
if [ "$(wc -l < "$LOG_FILE")" -gt "$MAX_LOG_ENTRIES" ]; then
        tail -n "$MAX_LOG_ENTRIES" "$LOG_FILE" > temp_log && mv temp_log "$LOG_FILE"
    fi
-----------------------------------BASH CODE SNIPPET-----------------------------------
```



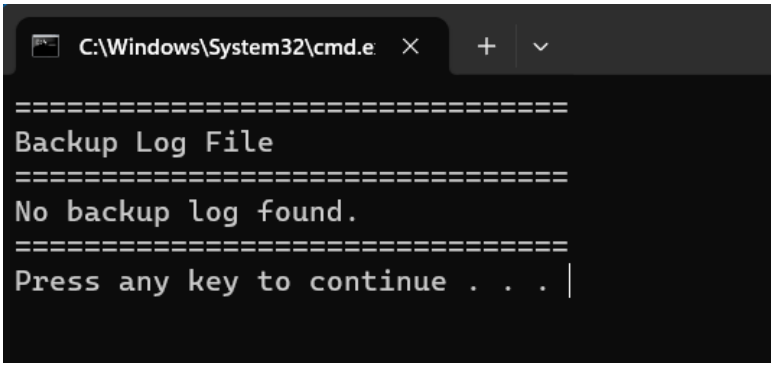
BASH Capture8- backup_log.txt saves five log entries.



BASH Capture 9- When it's the 6th one it deletes the oldest (1st) log entry and adds the 6th log as the 5th log.

6. **View log when no backups made**
   → Expected: Displays “No backup log found.

```
-----------------------------------MSDOS CODE SNIPPET-----------------------------------
:VIEW_LOG
CLS
ECHO ================================
ECHO Backup Log File
ECHO ================================
IF EXIST backup_log.txt (
    TYPE backup_log.txt
) ELSE (
    ECHO No backup log found.
)
ECHO ================================
PAUSE
GOTO MENU
-----------------------------------MSDOS CODE SNIPPET-----------------------------------
```

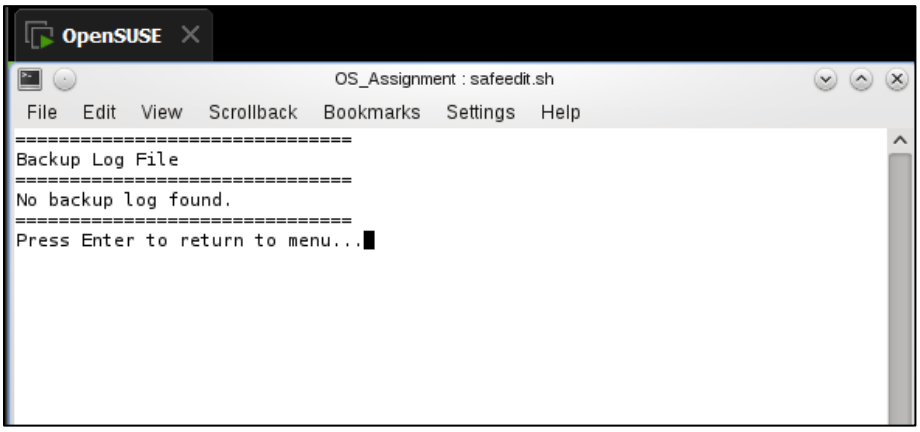


MSDOS Capture 13- Shows no backup log found message.

If the backup log does not exist, both scripts notify the user appropriately.

```
-----------------------------------BASH CODE SNIPPET-----------------------------------
View_log() {
    clear
    echo “==============================”
    echo “Backup Log File”
    echo “==============================”
    if [ -f “$LOG_FILE” ]; then
        cat “$LOG_FILE”
    else
        echo “No backup log found.”
    Fi
    echo “==============================”
    read -rp “Press Enter to return to menu…”
}
```

----------BASH CODE SNIPPET----------

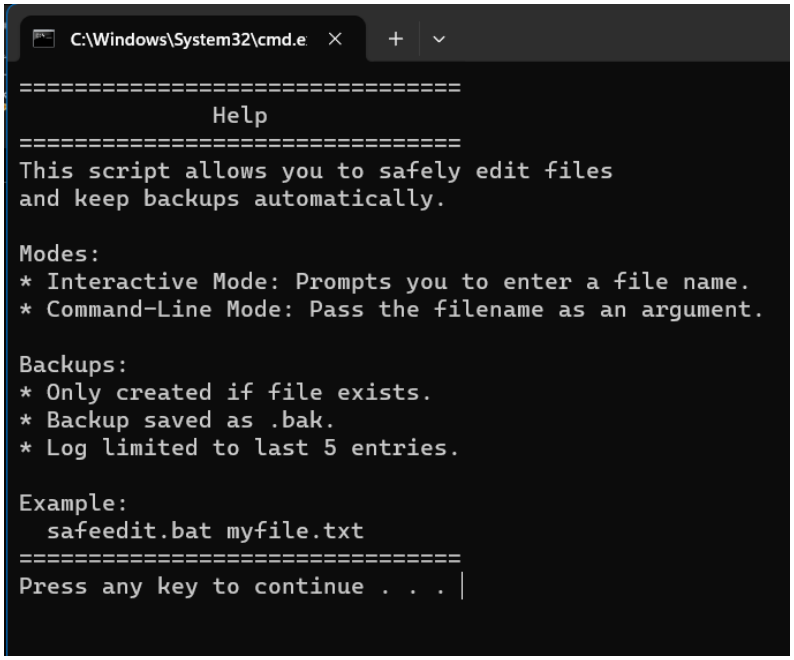


BASH Capture 10- Shows no backup log found message.

4.4 Help Function & Exit

7. **Use interactive menu option "3" for Help**
→ Expected: Displays help section and returns to menu.

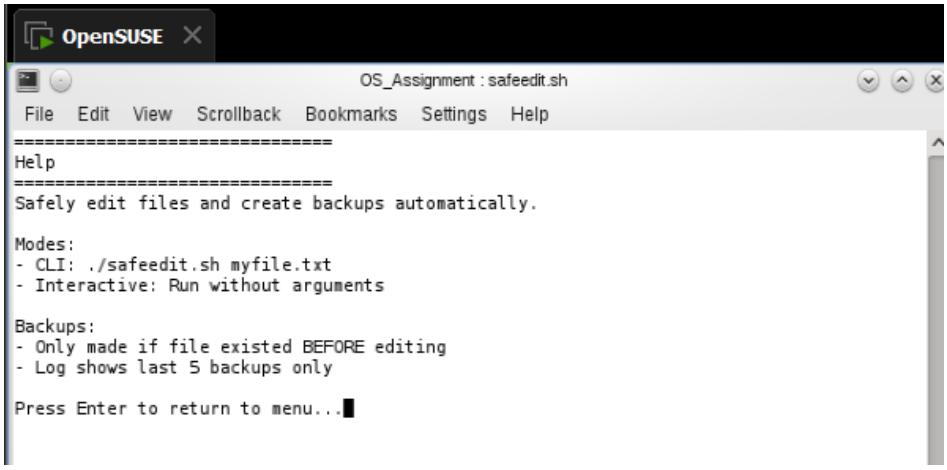
```
-----------------------------------MSDOS CODE SNIPPET-----------------------------------
IF "%CHOICE%"=="3" GOTO HELP
-----------------------------------MSDOS CODE SNIPPET-----------------------------------
```



MSDOS Capture 14- Shows help

The help option is accessible through the menu in both environments and offers guidance on script usage.

```
-----------------------------------BASH CODE SNIPPET-----------------------------------
3) show_help ;;
-----------------------------------BASH CODE SNIPPET-----------------------------------
```

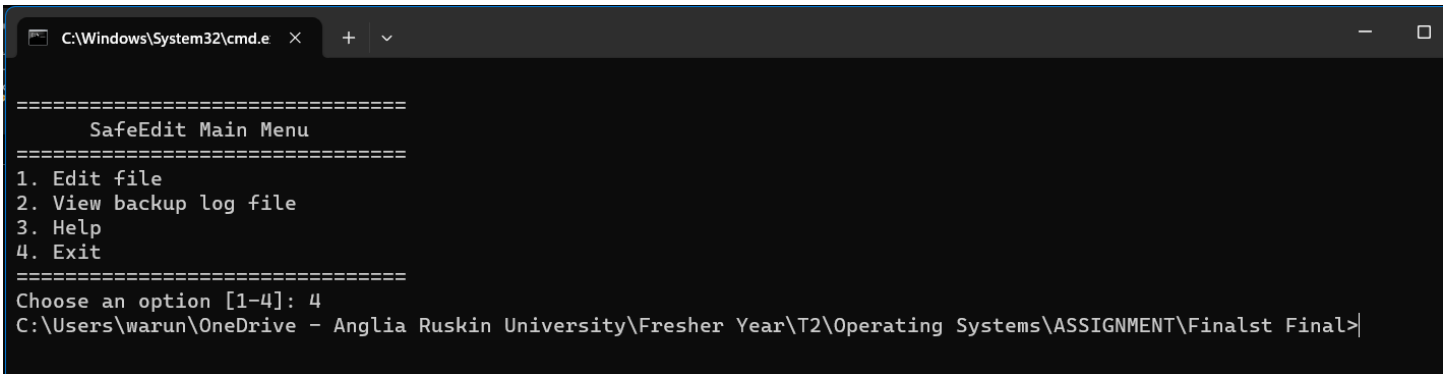


BASH Capture 11- Shows help

7. **Use interactive menu option "4" for Exit**
   → Expected: Closes the program.

```
-------------------------------------MSDOS CODE SNIPPET-------------------------------------
:EXIT
Endlocal
```

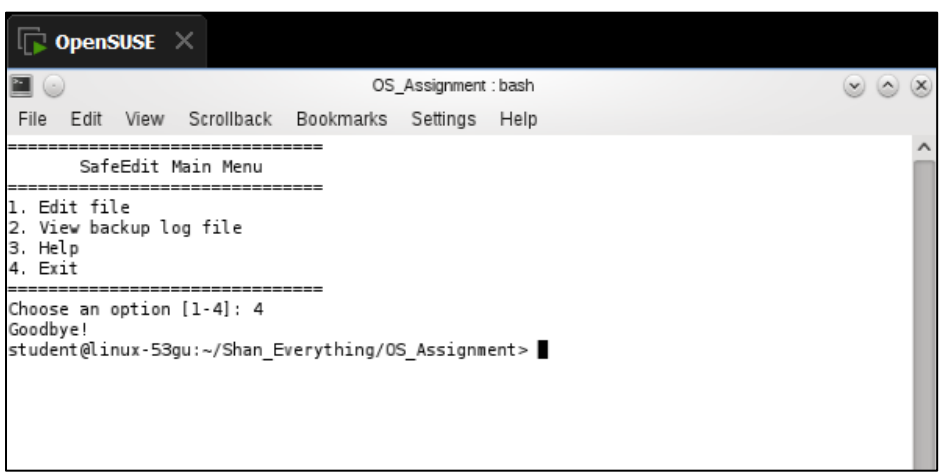
```
IF "%CHOICE%"=="4" GOTO EXIT
-------------------------------------MSDOS CODE SNIPPET-------------------------------------
```



MSDOS Capture15 – closes the program running.

I initially considered using just EXIT to close the program, but I found it wasn't very user-friendly as it would close the entire Command Prompt or terminal. To make it more user-friendly, I looked for alternative solutions, like using Endlocal (SuperUser, 2020) in MS-DOS and exit 0 in Bash(StackOverflow, 2012), which ensure only the script ends while keeping the terminal open .

```
-------------------------------------BASH CODE SNIPPET-------------------------------------
        4) echo "Goodbye!"; exit 0 ;;
-------------------------------------BASH CODE SNIPPET-------------------------------------
```

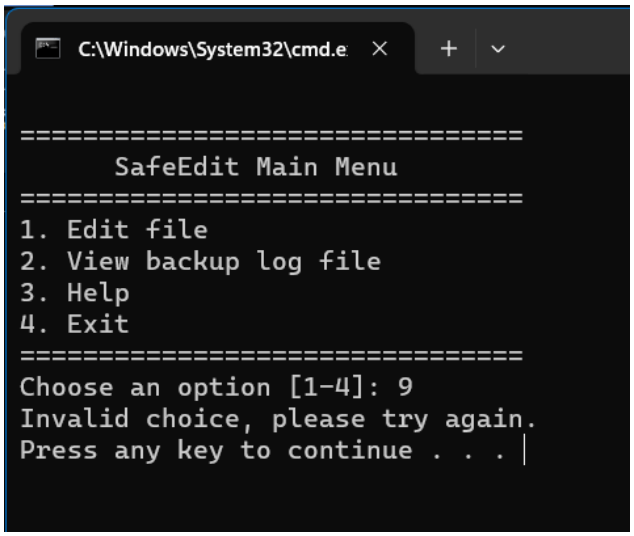


Bash Capture 12- Exit

4.5 Input Validation

8. **Enter invalid menu option (e.g., “9”)**
   → Expected: Displays error message and returns to menu.

```
-----------------------------------MSDOS CODE SNIPPET-----------------------------------
ECHO Invalid choice, please try again.
PAUSE
GOTO MENU
-----------------------------------MSDOS CODE SNIPPET-----------------------------------
```



MSDOS Capture 16 – when 9 was entered (an example option which is not 1-4) it shows the error message invalid choice and let’s try again.

Both scripts include input validation for menu selections and handle incorrect inputs gracefully.

```
-----------------------------------BASH CODE SNIPPET-----------------------------------
        *) echo "Invalid choice"; sleep 1 ;;
-----------------------------------BASH CODE SNIPPET-----------------------------------
```

5. Reference

Heath, J., 2015. *Linux Administration: A Beginner's Guide*. McGraw-Hill Education.

Microsoft, 2025. *wmic*. Available at: https://learn.microsoft.com/en-us/windows/win32/wmisdk/wmic [Accessed 28 March 2025].

Microsoft Docs, 2023. *Notepad in Windows*. Available at: https://learn.microsoft.com/en-us/windows/notepad/ [Accessed 28 March 2025].

Red Hat, 2022. *Using the vi editor*. Available at: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/using_basic_linux_tools/editing-text-files-using-vi_using-basic-linux-tools [Accessed 2 April 2025].

Stack Overflow, 2019. *Delete first few lines from a text file with Windows batch scripting*. Available at: https://stackoverflow.com/questions/57103963/delete-first-few-lines-from-a-text-file-with-windows-batch-scripting [Accessed 28 March 2025].

StackOverflow, 2012. *Any way to exit bash script, but not quitting the terminal*. Available at: https://stackoverflow.com/questions/9640660/any-way-to-exit-bash-script-but-not-quitting-the-terminal [Accessed 2 April 2025].

SuperUser, 2020. *Prevent "exit" command in script to close cmd*. Available at: https://superuser.com/questions/1636242/prevent-exit-command-in-script-to-close-cmd [Accessed 28 March 2025].

Tanenbaum, A.S., 2016. *Modern Operating Systems* (4th ed.). Pearson.

6. Appendix

6.1 MS-DOS Script (safeedit.bat)

```
@ECHO OFF
SETLOCAL ENABLEDELAYEDEXPANSION

REM Check for too many parameters
IF NOT "%~2"==""  (
    ECHO Too many parameters entered. Please provide only one filename.
    PAUSE
    EXIT /B
    )



REM checking for the help line thing
IF "%~1"=="-h" (
    GOTO HELP
)


REM If one parameter is passed, use it
IF NOT "%~1"==""  (
    SET FILE=%~1
    SET MODE=CLI
    GOTO EDIT_FILE
)

REM Interactive mode
:MENU
CLS
ECHO.
ECHO ================================
ECHO       SafeEdit Main Menu
ECHO ================================
ECHO 1. Edit file
ECHO 2. View backup log file
ECHO 3. Help
ECHO 4. Exit
ECHO ================================
SET /P CHOICE=Choose an option [1-4]:

IF "%CHOICE%"=="1" GOTO ASK_FILENAME
IF "%CHOICE%"=="2" GOTO VIEW_LOG
IF "%CHOICE%"=="3" GOTO HELP
IF "%CHOICE%"=="4" GOTO EXIT
```

```
ECHO Invalid choice, please try again.
PAUSE
GOTO MENU

:ASK_FILENAME
CLS
ECHO ================================
ECHO   Edit File (Interactive Mode)
ECHO ================================
ECHO What file do you wish to edit?
SET /P FILE=
SET MODE=INTERACTIVE
GOTO EDIT_FILE

:EDIT_FILE
CLS
ECHO ================================
ECHO        Editing: %FILE%
ECHO ================================

SET BACKUP_MADE=0

REM Check if file exists before editing
IF EXIST "%FILE%" (
    REM Create backup
    COPY /Y "%FILE%" "%FILE%.bak" >NUL
    SET BACKUP_MADE=1

    REM Log the backup with timestamp
    FOR /F "tokens=2 delims==" %%I IN ('WMIC OS GET LocalDateTime /VALUE') DO SET
DATETIME=%%I
    SET TIMESTAMP=!DATETIME:~0,4!-!DATETIME:~4,2!-!DATETIME:~6,2!
!DATETIME:~8,2!:!DATETIME:~10,2!:!DATETIME:~12,2!
    ECHO [!TIMESTAMP!] Backup created: %FILE% → %FILE%.bak >> backup_log.txt

    REM Trim log to last 5 entries
    FINDSTR /R /C:"Backup created:" backup_log.txt > temp_log.txt
    FOR /F "skip=5 delims=" %%L IN (temp_log.txt) DO (
        MORE +1 temp_log.txt > temp_log2.txt
        MOVE /Y temp_log2.txt temp_log.txt >NUL
    )
    MOVE /Y temp_log.txt backup_log.txt >NUL 2>NUL
) ELSE (
    REM File does not exist, create it but no backup
    ECHO File "%FILE%" does not exist. Creating a new file...
    ECHO. > "%FILE%"
)
```

```
REM Open the file for editing
NOTEPAD "%FILE%"

IF "%MODE%"=="CLI" (
    ECHO Done editing "%FILE%".
    GOTO EXIT
)
GOTO MENU

:VIEW_LOG
CLS
ECHO ================================
ECHO Backup Log File
ECHO ================================
IF EXIST backup_log.txt (
    TYPE backup_log.txt
) ELSE (
    ECHO No backup log found.
)
ECHO ================================
PAUSE
GOTO MENU

REM explains about the systems usage and what it will do
:HELP
CLS
ECHO ================================
ECHO               Help
ECHO ================================
ECHO This script allows you to safely edit files
ECHO and keep backups automatically.
ECHO.
ECHO Modes:
ECHO * Interactive Mode: Prompts you to enter a file name.
ECHO * Command-Line Mode: Pass the filename as an argument.
ECHO.
ECHO Backups:
ECHO * Only created if file exists.
ECHO * Backup saved as .bak.
ECHO * Log limited to last 5 entries.
ECHO.
ECHO Example:
ECHO   safeedit.bat myfile.txt
ECHO ================================
PAUSE
GOTO MENU
```

```
:EXIT
endlocal
```

6.2 Linux Script (safeedit.sh)

```
#!/bin/bash

LOG_FILE="backup_log.txt"
MAX_LOG_ENTRIES=5

log_backup() {
    local filename="$1"
    local timestamp
    timestamp=$(date +"[%Y-%m-%d %H:%M:%S]")
    echo "$timestamp Backup created: $filename → $filename.bak" >> "$LOG_FILE"

    if [ "$(wc -l < "$LOG_FILE")" -gt "$MAX_LOG_ENTRIES" ]; then
        tail -n "$MAX_LOG_ENTRIES" "$LOG_FILE" > temp_log && mv temp_log "$LOG_FILE"
    fi
}

edit_file() {
    local filename="$1"
    echo "================================"
    echo "Editing: $filename"
    echo "================================"

    FILE_EXISTED=0
    if [ -f "$filename" ]; then
        FILE_EXISTED=1
        cp -f "$filename" "$filename.bak"
    else
        touch "$filename"
        echo "File does not exist. Creating new file: $filename"
    fi

    vi "$filename"

    # After editing, if the file existed BEFORE, log the backup
    if [ "$FILE_EXISTED" -eq 1 ]; then
        log_backup "$filename"
    fi
}

view_log() {
    clear
    echo "================================"
    echo "Backup Log File"
    echo "================================"
    if [ -f "$LOG_FILE" ]; then
```

```
        cat "$LOG_FILE"
    else
        echo "No backup log found."
    fi
    echo "================================"
    read -rp "Press Enter to return to menu..."
}

show_help() {
    clear
    echo "================================"
    echo "Help"
    echo "================================"
    echo "Safely edit files and create backups automatically."
    echo
    echo "Modes:"
    echo "- CLI: ./safeedit.sh myfile.txt"
    echo "- Interactive: Run without arguments"
    echo
    echo "Backups:"
    echo "- Only made if file existed BEFORE editing"
    echo "- Log shows last 5 backups only"
    echo
    read -rp "Press Enter to return to menu..."
}

if [ $# -gt 1 ]; then
    echo "Too many parameters. Provide only one filename."
    exit 1
elif [ $# -eq 1 ]; then
    edit_file "$1"
    exit 0
fi

while true; do
    clear
    echo "================================"
    echo "       SafeEdit Main Menu      "
    echo "================================"
    echo "1. Edit file"
    echo "2. View backup log file"
    echo "3. Help"
    echo "4. Exit"
    echo "================================"
    read -rp "Choose an option [1-4]: " choice

    case "$choice" in
```

```
        1)
            clear
            echo "==============================="
            echo "Edit File (Interactive Mode)"
            echo "==============================="
            read -rp "Enter filename: " filename
            edit_file "$filename"
            ;;
        2) view_log ;;
        3) show_help ;;
        4) echo "Goodbye!"; exit 0 ;;
        *) echo "Invalid choice"; sleep 1 ;;
    esac
done
```