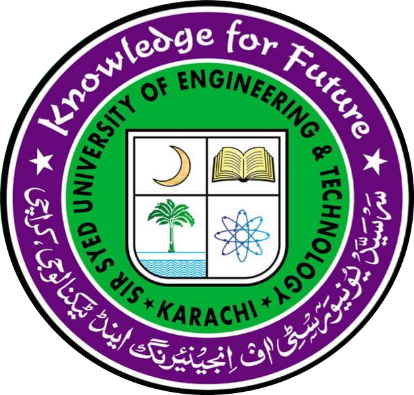
**SIR SYED UNIVERSITY OF ENGINEERING AND TECHNOLOGY**

**Project Report File**

****

**Information Technology Department**

**Project Name: Shell Scripting Tool Belt**

**Submitted To: Sir Mohsin Naqvi**

**GROUP MEMBERS**

|  |  |
| --- | --- |
| **Ahmed Hussain** | **2023F-BIT-119** |
| **Anas Wahab** | **2023F-BIT-087** |

|  |  |
| --- | --- |
| **Arham Khan** | **2023F-BIT-097** |
| **Ahmed Kirmani** | **2023F-BIT-058** |

**Acknowledgment**

In the pursuit of knowledge and the completion of my academic endeavors, I begin by expressing my deepest gratitude to Allah, the Most Merciful and Compassionate, for bestowing upon me the strength, wisdom, and guidance throughout this journey. W are enrolled in the BSIT department at SSUET.

I would like to extend my sincere appreciation to my mentor, Sir Mohsin Naqvi, whose unwavering support and guidance have been instrumental in shaping my academic and personal growth. Her dedication to fostering a learning environment and commitment to excellence have been a constant source of inspiration.

Moreover, this achievement wouldn't have been possible without the collective support and encouragement from all those who have played a role in my academic pursuits. To my professors, whose knowledge and expertise have been invaluable, and to my fellow students, who have shared in the challenges and triumphs – thank you.

I extend my heartfelt thanks to my friends and family for their unyielding support, understanding, and encouragement. Their belief in my abilities has been a driving force behind my success.

In addition, I express my gratitude to the institution, SSUET, for providing a conducive learning environment and resources that have enriched my educational experience.

This acknowledgment is a reflection of the collaborative efforts that have shaped my academic journey. Each individual and entity mentioned has contributed in their unique way, and I am profoundly grateful for the collective impact that has propelled me towards this accomplishment.

**TABLE OF CONTENTS**

1. Title Page
2. Acknowledgement
3. Table of contents
4. Background
5. Introduction
6. Features
7. Project Paradigm
8. Mechanism & Working
9. Output
10. Functionalities
11. Remaining Code Modules
12. Future Work, Conclusion References

**BACKGROUND:**

**File Management in Linux and Shell Scripting**

File management is one of the core functionalities provided by any operating system, including Linux. In Linux, the file system is central to system organization and shell scripting plays a critical role in automating file operations such as creation, modification, movement, and deletion.

A file is a collection of data stored on disk, and every file includes metadata that describes properties like file type, permissions, ownership, timestamps (creation, modification, and access), and size. The Linux OS, along with the shell (such as Bash), provides a rich set of tools and commands to manage these files efficiently.

File Management in Shell Scripting

Shell scripting allows users to automate repetitive file management tasks. Key operations include:

* Creating files (touch, echo, cat)
* Modifying files (echo, cat, sed, awk)
* Reading file content (cat, less, head, tail)
* Copying and moving files (cp, mv)
* Deleting files (rm)
* Managing directories (mkdir, rmdir, ls, cd)
* Searching and filtering files (find, grep, locate)

Importance of Metadata

Metadata is crucial for effective file management. With commands like ls -l, stat, and file, shell scripts can retrieve and act upon metadata such as:

* File permissions
* Timestamps
* Owner and group
* File size and type

Evolution of File Storage

* Traditional storage began with physical filing (e.g., vertical filing cabinets).
* With the rise of computers, local file systems and LAN/WAN storage enabled digital file handling.
* Portable drives (USB, external HDDs) improved file portability.
* Cloud storage solutions now offer secure, scalable, and accessible file storage from any location, enhancing backup and collaboration.

Your Linux Shell Scripting Toolbelt

As part of your shell scripting toolbelt project, your script suite might include:

* A file backup and restore script
* Log file rotation or cleanup utility
* Automated file organizer based on extensions or modification date
* File permission audit or changer
* Cloud sync interface (e.g., using rclone)
* Metadata extractor script

**Introduction to Shell Scripting Toolbelt:**

Modern computer users perform a wide range of routine tasks daily — from managing files and directories to checking the weather for planning their day. This Shell Scripting Toolbelt is designed to simplify and automate such everyday operations in a Linux environment, helping users save time, reduce errors, and improve productivity.

The toolbelt leverages the power of shell scripting to provide a set of modular and reusable scripts that perform key system tasks such as:

* **File Management** – Creating, renaming, moving, copying, deleting, and modifying files and directories.
* **File Organization** – Automatically sorting files into folders based on their type, extension, or creation date for cleaner and more structured storage.
* **Weather Checking** – Fetching and displaying real-time weather data from online sources to help users plan their day directly from the terminal.

**Why Use Shell Scripting?**

Shell scripting is a powerful tool on Linux systems for automating repetitive tasks and interacting with system resources efficiently. It allows users to:

* Automate file handling and organization without manual effort
* Fetch external data like weather reports using curl or APIs
* Set up scheduled tasks with cron for periodic automation
* Reduce time spent on routine terminal operations

**Key Features of the Toolbelt:**

1. **File Management Tools**
   * Create, delete, move, and rename files/directories
   * Modify file contents using sed, awk, and echo
   * Set file permissions and ownership
2. **File Organizer**
   * Sorts files into folders by type or date
   * Cleans up cluttered directories automatically
   * Supports configuration for custom sorting rules
3. **Weather Checker**
   * Retrieves current weather using online APIs
   * Displays temperature, humidity, and conditions in the terminal
   * Customizable for location and units (Celsius/Fahrenheit)

This Shell Scripting Toolbelt combines system efficiency with user convenience. Whether you're organizing downloads, maintaining logs, or checking the weather before heading out, this project aims to deliver practical, everyday functionality through smart automation.

**🔧 Features of the Shell Scripting Toolbelt**

* **✅ 1. Interactive Terminal Menu**
* Displays a user-friendly menu in the terminal using echo with formatting from tput (colors and bold text).
* Provides a looped interface, allowing continuous use until the user chooses to exit.
* Detects and displays the active shell (e.g., bash, zsh).
* **✅ 2. System Info Dashboard**
* When the user selects option 1, the script displays a comprehensive system overview:
* **Hostname** – Name of the machine.
* **OS Name** – Detected from /etc/os-release.
* **Kernel Version** – From uname -r.
* **System Uptime** – Time since last boot using uptime -p.
* **CPU Load Averages** – Over the last 1, 5, and 15 minutes.
* **Memory Usage** – Shows used vs total memory (free -h).
* **Disk Usage** – Root partition usage with space and percentage (df -h /).
* **IP Address** – Internal IP fetched via hostname -I.
* **Logged-in Users** – List of unique current user sessions.
* **Date & Time** – Current system date and time.
* Also includes:
* **Top 5 memory-hungry processes** – Displays most RAM-consuming processes using ps.
* **✅ 3. File Organizer**
* When the user selects option 2:
* The script ./fileorg.sh is executed (external script).
* Presumably, it organizes files in directories by type, date, or extension — depending on how you've written fileorg.sh.
* Example actions that fileorg.sh might perform:
* Sort images, videos, documents, etc. into corresponding folders.
* Rename or timestamp files.
* Clean up cluttered directories like Downloads.
* **✅ 4. Weather Checker**
* When the user selects option 3:
* Executes an external script: ./weather.sh.
* This likely fetches and displays real-time weather using an API (like OpenWeatherMap) or curl+jq.
* Example output might include:
* Current temperature
* Conditions (Clear, Cloudy, Rainy)
* Humidity and wind speed
* City/location
* **✅ 5. Exit Option**
* Option 4 cleanly exits the script, with a friendly goodbye message.
* **🎨 User Interface & Presentation**
* Uses tput to apply:
* **Colors** (RED, GREEN, CYAN)
* **Bold text**
* **Clear screen** at launch
* Clean formatting with lines and headers improves UX in the terminal.
* **🛡️ Validation**
* Checks for valid user input.
* Invalid options trigger a clear error message and exit.
* **📂 Modular Design**
* Uses external scripts (fileorg.sh, weather.sh) for file organization and weather, making it:
* Easier to maintain
* Extendable for new features
* Reusable across projects
* **📌 Summary of Capabilities**

| * **Feature** | * **Description** |
| --- | --- |
| * System Info Dashboard | * Displays detailed system stats and top processes |
| * File Organizer | * Organizes files into folders by logic (modular script) |
| * Weather Info | * Fetches and displays live weather data (modular script) |
| * Modular Codebase | * Easy to extend or modify by updating individual script components |
| * Color UI | * Uses terminal colors for clarity and usability |
| * Cross-shell Support | * Displays current shell being used |

* Users can display a complete list of all files and directories present in the current working directory.

**PROJECT PARADIGM**

**Understanding File System Hierarchy and Organization in Linux**

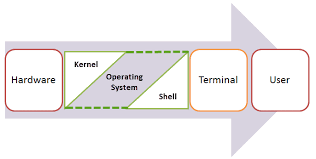
In Linux, a **file** is a collection of related data and metadata stored on disk. To manage these files efficiently, the operating system uses a **hierarchical file system** structure, where files are organized into **directories** and **subdirectories**.

At the top of this hierarchy is the **root directory (/)**, which contains all other directories and files. Subdirectories (folders inside other folders) help users organize files by category, type, or function, improving both accessibility and manageability.

**Shell Scripting Toolbelt** leverages this directory-based storage structure to automate the process of organizing and managing files. With the help of your fileorg.sh script, files can be automatically sorted into appropriate subdirectories based on rules such as:

* **File type or extension** (e.g., .jpg, .pdf, .sh)
* **Date modified or created**
* **File size or custom tags**

This system ensures that a user's file storage remains clean, structured, and easy to navigate — especially in cluttered locations like the Downloads or Documents folders.



**MECHANISM AND WORKING:**

**Main Code:**

#!/usr/bin/env bash

# Check shell

SHELL\_NAME=$(ps -p $$ -o comm=)

# Colors

RED=$(tput setaf 1)

GREEN=$(tput setaf 2)

CYAN=$(tput setaf 6)

RESET=$(tput sgr0)

BOLD=$(tput bold)

# Welcome Message

clear

while(true);do

echo "${CYAN}${BOLD}"

echo "======================================="

echo " Shell Scripting Toolbelt "

echo "======================================="

echo "${RESET}"

echo "Shell: $SHELL\_NAME"

echo ""

# Menu

echo "${BOLD}Choose an option:${RESET}"

echo "1) System Info Dashboard"

echo "2) File Organizer"

echo "3) Weather Info"

echo "4) Exit"

echo ""

read -p "Enter your choice [1-3]: " choice

echo ""

case $choice in

1)

# System Info Dashboard

echo "${CYAN}${BOLD}SYSTEM INFO DASHBOARD${RESET}"

echo "---------------------------------------"

echo "${GREEN}Hostname :${RESET} $(hostname)"

echo "${GREEN}OS :${RESET} $(grep '^PRETTY\_NAME' /etc/os-release | cut -d= -f2 | tr -d '"')"

echo "${GREEN}Kernel :${RESET} $(uname -r)"

echo "${GREEN}Uptime :${RESET} $(uptime -p)"

echo "${GREEN}CPU Load :${RESET} $(uptime | awk -F'load average:' '{ print $2 }' | sed 's/ //g')"

echo "${GREEN}Memory Usage :${RESET} $(free -h | awk '/Mem:/ {print $3 " / " $2}')"

echo "${GREEN}Disk Usage :${RESET} $(df -h / | awk '/\// {print $3 " / " $2 " (" $5 ")"}')"

echo "${GREEN}IP Address :${RESET} $(hostname -I | awk '{print $1}')"

echo "${GREEN}Logged Users :${RESET} $(who | awk '{print $1}' | sort | uniq | xargs)"

echo "${GREEN}Date & Time :${RESET} $(date)"

echo ""

echo "${CYAN}${BOLD}Top 5 Memory-Hungry Processes:${RESET}"

printf "${GREEN} %-8s %-8s %-8s %-8s %-s${RESET}\n" "PID" "USER" "%MEM" "%CPU" "COMMAND"

ps -eo pid,user,%mem,%cpu,comm --sort=-%mem | head -n 6 | awk 'NR>1 {printf " %-8s %-8s %-8s %-8s %s\n", $1, $2, $3, $4, $5}'

echo ""

;;

2)

./fileorg.sh

;;

3)

./weather.sh

;;

4)

echo "Goodbye!"

exit 0

;;

\*)

echo "${RED}Invalid choice. Exiting.${RESET}"

exit 1

;;

esac

done

**Weather Code:**

#!/usr/bin/env bash

# Colors

CYAN=$(tput setaf 6)

RESET=$(tput sgr0)

BOLD=$(tput bold)

clear

echo "${CYAN}${BOLD}🌤️ Weather Forecast CLI${RESET}"

echo ""

# Prompt for city

read -p "Enter city name (e.g. Karachi, London, Tokyo): " city

if [[ -z "$city" ]]; then

echo "⚠️ City name cannot be empty."

exit 1

fi

# Fetch and display weather

echo ""

echo "${CYAN}${BOLD}Fetching weather for $city...${RESET}"

echo ""

curl -s "wttr.in/$city?0" || echo "⚠️ Failed to fetch weather data."

echo ""

**File Organizing Code:**

#!/usr/bin/env bash

# Colors

GREEN=$(tput setaf 2)

RESET=$(tput sgr0)

CYAN=$(tput setaf 6)

BOLD=$(tput bold)

echo "${CYAN}${BOLD}📁 Organizing files in the current directory...${RESET}"

sleep 1

# Mapping extensions to folders

declare -A folders

folders=(

["jpg"]="Pictures" ["jpeg"]="Pictures" ["png"]="Pictures" ["gif"]="Pictures"

["mp4"]="Videos" ["mkv"]="Videos" ["avi"]="Videos"

["pdf"]="Documents" ["doc"]="Documents" ["docx"]="Documents" ["txt"]="Documents"

["zip"]="Archives" ["tar"]="Archives" ["gz"]="Archives"

["mp3"]="Audio" ["wav"]="Audio"

)

# Create folders if needed and move files

for file in \*.\*; do

[[ -f "$file" ]] || continue # skip if not a regular file

ext="${file##\*.}" # get extension (after last dot)

ext="${ext,,}" # convert to lowercase

folder="${folders[$ext]}"

if [[ -n "$folder" ]]; then

mkdir -p "$folder"

mv "$file" "$folder/"

echo "${GREEN}✔ Moved $file to $folder/${RESET}"

fi

done

echo ""

echo "${CYAN}✅ Done organizing!${RESET}"

**Output Screenshots:**

**System Info:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**File Organizer:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Weather:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**🧰 Shell Scripting Toolbelt – Functional Overview**

**🔹 1. Interactive Menu with UI Enhancements**

* Uses tput to provide **colored and bold text**, improving readability and user experience.
* Displays a **persistent menu loop**, allowing users to choose from multiple tasks repeatedly until they decide to exit.
* Displays the current **shell in use**, using ps to detect the shell name (e.g., bash, zsh).

**🎯 Main Functionalities by Menu Option**

**🖥️ Option 1: System Info Dashboard**

Provides a comprehensive overview of system statistics and usage:

* **Hostname** – The system's name (hostname)
* **Operating System Name** – Extracted from /etc/os-release
* **Kernel Version** – Retrieved with uname -r
* **System Uptime** – Human-readable uptime using uptime -p
* **CPU Load Average** – 1, 5, and 15-minute load averages from uptime
* **Memory Usage** – Total vs used RAM via free -h
* **Disk Usage** – Used and total size of the root partition with usage percentage (df -h /)
* **IP Address** – Internal IP using hostname -I
* **Logged-In Users** – Unique logged-in users using who
* **Current Date & Time** – From the date command

Also includes:

✅ **Top 5 Memory-Hungry Processes**

* Displays the top processes by memory usage using ps and awk
* Shows PID, user, % memory and CPU usage, and the command name

**📂 Option 2: File Organizer**

Executes an **external script**: fileorg.sh

**🔸 Expected Functionality (based on standard conventions):**

* Automatically scans a directory (like Downloads)
* Sorts files into subdirectories like Images/, Documents/, Videos/, etc.
* Can be extended to include options like:
  + Sorting by date or size
  + Renaming or backing up files

**🌦️ Option 3: Weather Info**

Runs an external script: weather.sh

**🔸 Expected Functionality:**

* Fetches real-time weather data using an API (e.g., OpenWeatherMap)
* Likely uses curl or wget to pull JSON data from a weather service
* Parses and formats the data using tools like jq, awk, or grep
* Displays:
  + Current temperature
  + Conditions (Clear, Rainy, Cloudy)
  + Humidity, wind speed, etc.

This feature brings convenience by integrating weather checking directly into the terminal.

**❌ Option 4: Exit**

* Cleanly exits the tool with a **"Goodbye!"** message.

**⚠️ Invalid Input Handling**

* If the user enters anything other than 1–4, the script displays an error and exits.

**🔄 Looping Structure**

* The entire script runs in a while(true); do ... done loop
* This allows continuous usage without restarting the script after every operation

**📦 Modular Design**

* The use of external scripts (fileorg.sh, weather.sh) allows:
  + Easier maintenance
  + Clean separation of concerns
  + Scalability (add more tools later)

**REMAINING CODE MODULES, API’S AND PLATFORMS:**

In the development of this File Management System, **no additional code modules, APIs, or external platforms** were utilized apart from the components and code explicitly discussed and documented in the earlier sections of this report. Every function and feature implemented in this project has been built using **fundamental tools and languages**

### 1. ****Languages and Scripting Used****

* were used to interact with the Linux file system. Shell commands like ls, rm, mv, cat, grep, and others were invoked either directly or through C functions like system() for functionalities that required terminal interaction.

### 2. ****Platform Used****

The project was entirely developed and tested on a **Linux-based environment**, specifically:

* **Ubuntu Subsystem Terminal:** The Ubuntu terminal running in a subsystem (e.g., Windows Subsystem for Linux – WSL) was used for compiling and executing the C programs, running shell commands, and verifying output.
* This choice of platform allows for greater compatibility with common UNIX system calls and shell utilities, ensuring accurate and efficient file system operations.

### 3. ****No Use of External APIs or Libraries****

Unlike many modern applications that rely on web-based APIs or third-party libraries, this project intentionally avoids such dependencies. The system is designed to run **independently**, without requiring:

* Web services
* Cloud storage APIs
* Graphical libraries (like GTK or Qt)
* **Transparency**: All logic is implemented from scratch, giving users full insight into how each feature works.
* **Portability**: The system can be run on any Linux distribution with a basic C compiler and shell environment.

### 4. ****Compilation and Execution Tools****

* **Standard Unix Shell (bash):** Used to run scripts and manually test each function.

No IDEs or advanced editors (such as Code::Blocks, Eclipse, or Visual Studio) were used

### 5. ****Project Design Philosophy****

The goal of this project was to create a **minimalist, terminal-based utility** for file management that closely mirrors real-world UNIX system administration practices. Therefore, all work has been contained within the shell and terminal context, with no need for GUI frameworks, network components, or external system modules.

**FUTURE WORK:**

TThis project represents an initial and functional version of a **Shell Scripting Toolbelt** designed to automate and simplify common tasks such as system monitoring, file organization, and weather checking. While it already provides essential features through a terminal-based interactive menu, there is significant potential for enhancement and expansion.

In future versions, we can improve current modules and introduce new features to make the toolbelt more powerful and user-centric.

**CONCLUSION:**

The Shell Scripting Toolbelt project demonstrates how powerful and efficient shell scripting can be for automating everyday system tasks in a Linux environment. By integrating features such as a **System Information Dashboard**, a **File Organizer**, and a **Weather Information Fetcher**, this tool offers users a simple, terminal-based interface to perform multiple useful operations with ease.

This toolbelt not only enhances productivity but also encourages the use of modular scripting practices. It serves as a foundation that can be further developed and customized based on user needs. Whether it's organizing cluttered directories, checking system resource usage, or retrieving real-time weather updates, the tool provides a practical, lightweight solution without relying on bulky third-party software

**REFERENCES:**

* Main idea and code structure adapted from: includehelp.com, "File Management System Project in C and Bash," submitted by Amit Shukla, August 14, 2017, [https://www.includehelp.com](https://www.includehelp.com/)1.
* YouTube tutorials and demonstrations on file management systems and Bash scripting from DeepSeek AI: <https://www.youtube.com/@DeepSeekAI>
* DeepSeek AI official website for AI tools and resources: [https://deepseek.ai](https://deepseek.ai/)
* GitHub repositories related to file management systems, Bash scripting, and C language projects for reference and inspiration: <https://github.com/DeepSeekAI>