# Project Based Evaluation

Project Report
Semester-IV (Batch 2023)

User & Group Manager Tool with Session Logger

**Supervised By:**
Mrs. Parul Gehlot

**Submitted By:**
Tanishq Gupta, 2310992384(G19)
Ansh Jolly, 2310991694(G19)
Armaan Singh Thind, 2310991704(G19)
Krishna Saxena, 2310991705(G19)

**Department of Computer Science and Engineering Chitkara University Institute of Engineering & Technology, Chitkara University, Punjab**

# Table of Contents

# 1. Introduction

User account management is a critical aspect of Linux system administration, providing controlled and auditable access to system resources. In large or multi-user environments, manual account handling can be tedious and error-prone, making automation desirable. Automating these tasks via a Terminal User Interface (TUI) improves efficiency, reduces human error, and enforces consistent policies. Terminal-based dialogs (using dialog) allow administrators to perform complex operations through intuitive, text-based menus. This project implements a Bash script providing a dialog-driven TUI for Linux user and group management, enabling root users to create, modify, and delete accounts interactively.

## 1.1 Background and Motivation

Modern Linux systems use user and group accounts for security and resource management. Each account corresponds to a record in system files like /etc/passwd and /etc/group, which define user identities, home directories, shells, and permissions. Managing these accounts typically involves commands like adduser, usermod, and groupadd from the command line. However, typing commands repeatedly for routine tasks can be slow and error-prone, especially for administrators managing many accounts. A TUI script makes these functions more accessible by providing guided menus and input validation. As one commentator noted, dialog-based interfaces can “build a very nice and simple UI that will work perfectly on your terminal”, reducing manual keystrokes and mistakes.

## 1.2 Objectives of the Project

The primary objective is to create an interactive, menu-driven Bash script for Linux root users to manage accounts. The script should allow creating, modifying, and deleting user and group accounts, including password and group membership management. It should validate user input (e.g. enforcing username format and length), enforce root-only execution, and maintain detailed logs of all account changes and user logins for auditing.

In summary, the aims are to automate user administration tasks via a dialog TUI and to implement robust logging and validation features.

## 1.3 Scope of the Work

This project focuses on local account management on a single Linux system. It covers standard account-related operations such as adding/removing users and groups, changing passwords, and updating group memberships. The script enforces security measures like root privileges and input checks. It does *not* integrate with network directory services or provide a graphical GUI. Instead, it emphasizes reliability and completeness of fundamental tasks. Logging of user logins and administrative actions is included, but advanced network monitoring or distributed account features are beyond the scope.

## 1.4 Report Structure

This report is organized into several sections. Section 2 describes the system environment, including required hardware, software, and tools. Section 3 overviews key concepts (user accounts, groups, PAM, log files) and system components like /etc/passwd. Section 4 sketches UML diagrams for use cases and flows. Section 5 details the implementation: configuring the environment and walking through the script's logic and commands. Section 6 discusses security hardening, performance, and backup considerations. Section 7 covers testing: scenarios, expected outcomes, and troubleshooting. Section 8 reflects on challenges encountered. Section 9 concludes with accomplishments and future work. References are listed in Section 10, and appendices (Section 11) include configuration file snippets and script excerpts.

# 2. System Environment

## 2.1 Hardware and Software Requirements

This user management TUI was developed to run on standard x86_64 Linux servers or desktops. Typical hardware requirements are minimal: any modern CPU, a few gigabytes of RAM, and storage for the operating system and log files. For example, a test environment could be a virtual machine with 1–2 GB RAM and 10 GB disk. The script is written in Bash and relies on core GNU utilities (like grep, cut, awk) and the dialog package for the interface. It also requires common user/group commands (adduser/useradd, usermod, groupadd, passwd/chpasswd), which are usually included in most Linux distributions. If dialog is missing, the script will prompt to install it (e.g. sudo apt-get install dialog on Debian/Ubuntu or sudo yum install dialog on RHEL/CentOS).

Software dependencies include a Bash shell (tested on Bash 5.x) and the dialog utility. The script checks for root privileges at startup, so it must be run as root or via sudo. It writes log files under /var/log/user_activity, so the system's logging directories must be writable by root. No special libraries or external binaries (beyond standard system tools) are needed.

## 2.2 Linux Distribution and Version

The implementation is distribution-agnostic and was tested on both Debian/Ubuntu (e.g. Ubuntu 22.04 LTS) and Red Hat/CentOS (RHEL 8). The script uses only POSIX-compliant shell features and common utilities, so it should run on any mainstream Linux distro. The dialog interface works similarly on both Debian-based and RPM-based systems. For package installation hints, the script comments mention apt-get for Debian/Ubuntu and yum for RHEL/CentOS, making it clear which package manager to use for dialog. It has been verified on a Debian 11 system and on CentOS 8 without modification.

## 2.3 Tools and Utilities Used

The project relies on standard system administration tools and libraries:

- Bash Shell: The script is written for the Bourne Again Shell, using functions, conditionals, loops, and built-in string operations. Bash is the default shell on most Linux systems.

- Dialog: The text-based UI is implemented with dialog. These programs provide interactive text boxes, menus, checklists, and prompts.

- User/Group Commands: Commands like useradd/adduser, usermod, groupadd, groupmod, groupdel, userdel, passwd or chpasswd, and id are used to perform system modifications. For example, useradd creates a new account, and chpasswd batch-updates passwords from username:password input.

- Utilities: The script employs tools like grep, cut, awk, and sed to parse system files (/etc/passwd, /etc/group) for generating lists or extracting fields. For instance, it uses getent passwd | cut -d: -f1 to list usernames. The getent command is used to query system account databases, which is safer for configurations like NIS or LDAP (if present), though this script focuses on local entries.

- PAM (Pluggable Authentication Modules): While not a utility per se, PAM configuration files (/etc/pam.d/login, /etc/pam.d/sshd, etc.) are modified by the script to enable login/logout logging. The pam_exec.so module is used to run a custom logging script at session open/close.

- Logging Tools: Standard file creation (touch) and permission (chmod) commands are used to set up log files under /var/log/user_activity. No third-party logging library is needed.

Overall, the script assembles these tools to create an interactive management interface. No new binaries are written; the script orchestrates existing commands through Bash and dialog.

# 3. Conceptual Overview

## 3.1 Key Concepts Related to the Project

At its core, the project manages *user accounts* and *groups* on Linux. A user account on Linux is a unique identity for a person or process; it has attributes such as a username, user ID (UID), primary group ID (GID), home directory, and default shell. Each account is represented by a line in /etc/passwd (username, UID, GID, GECOS field, home, shell) and an entry in /etc/shadow for the encrypted password. Linux distinguishes between *regular users* (day-to-day accounts) and *administrative (root) accounts* with unrestricted privileges. Groups are collections of users that share certain permissions; group membership is listed in /etc/group.

Security is a key concept: user accounts provide authentication and access control. For example, normal users have limited rights, while the root user is all-powerful. Effective user management includes ensuring that only valid accounts exist and that their privileges are appropriate. The script enforces a strict naming convention for usernames to avoid unusual or unsafe names. By default, Debian/Ubuntu's adduser requires a username to match the regex ^[a-z][-a-z0-9_]*$, meaning it must start with a lowercase letter and contain only lowercase letters, digits, hyphens, or underscores. This script's validation is similar, ensuring no empty or overly long names (max 32 chars) or invalid characters are used.

Another key concept is terminal-based user interfaces (TUI). A TUI provides dialog boxes and menus within the terminal, as opposed to purely textual commands or a full graphical interface. Tools like dialog enable TUIs in shell scripts, presenting menus, input prompts, and messages to the user. For example, the script uses dialog --menu to present a list of options, and dialog --inputbox to ask for text input. The benefit of a TUI is that it guides the administrator through tasks step-by-step, reducing the need to memorize flags or type repetitive commands.

Logging and auditing is also central. Best practices dictate that administrative actions should be logged for accountability. This script logs two types of information: (a) account changes, such as creating or deleting a user, and (b) user session events, i.e. logins and logouts. The account-change log captures the admin's username, action, target account, and details with timestamps. Login/logout logging is implemented via PAM: a script (log_login.sh) is invoked on each user session open/close, writing to a log file. This way, all login and logout times (and remote hosts, TTY, etc.) are recorded. These practices align with security auditing recommendations.

## 3.2 Relevant System Components and Files

Several system files and components are directly involved:

- /etc/passwd: Stores basic user information (username, UID, GID, home, shell) in a colon-separated line per user. This script reads and manipulates /etc/passwd indirectly via commands like getent passwd and useradd. Only root can modify it.

- /etc/shadow: Contains encrypted passwords and password aging info. While /etc/shadow itself is not directly edited by the script, the passwd and chpasswd commands update this file when setting passwords.

- /etc/group: Lists group names, GIDs, and members. The script queries and modifies it via getent group, groupadd, groupmod, and by editing group memberships.

- /etc/gshadow: Secure group password file (for group administration). Not explicitly touched by this script (common group tools handle it).

- PAM Configuration: /etc/pam.d/login and /etc/pam.d/sshd define authentication session rules. The script appends lines like session optional pam_exec.so /etc/security/log_login.sh to these files, ensuring the custom log_login.sh runs whenever a user logs in (console or SSH).

- /etc/profile and /etc/bash.bash_logout: These shell startup files are updated to call log_login.sh on login/logout. For example, /etc/profile is given lines to invoke the logging script with PAM_TYPE=open_session on user login, and /etc/bash.bash_logout is configured to call it with PAM_TYPE=close_session at logout.

- /var/log/user_activity/: A dedicated directory created by the script. It contains two files: user_login.log (recording each login/logout event with user, time, service, etc.) and account_changes.log (recording admin actions on accounts). The script ensures this directory (mode 750) and files (mode 640) exist, to keep logs secure.

- /etc/adduser.conf: (Ubuntu-specific) Contains settings like NAME_REGEX for valid usernames. The script's username validation is consistent with these standards.

- System Utilities: Commands like id, whoami, groups, useradd, groupadd, usermod, groupmod, userdel, groupdel and others interact with system databases. For example, id username checks for existence of a user. The whoami or id -un command determines the admin username for logging. Utilities like date provide timestamps for log entries.

These components together allow the script to interface with the underlying system. The script assumes a standard Linux account system; it does not interact with directory services or unusual authentication modules beyond PAM. It relies on the conventional structure of account files and standard commands to do its work.

## 3.3 Linux Commands and Services Involved

The script uses a variety of built-in services and commands:

- dialog: As noted, these create the text-based UI. The script uses options like --inputbox, --yesno, --menu, --checklist, and --msgbox to collect input and display messages. Each dialog command returns a status code and selected text, which the script captures to proceed.

- Account commands:

  - useradd or adduser: Creates new user accounts. (adduser on Debian is a friendlier wrapper around useradd.) The script typically calls useradd with flags like -m (create home directory), -d /home/$user, -s /bin/bash, and -c to set the comment/full name.

  - usermod: Modifies existing users (changing usernames, home directories, etc.). For example, usermod -l newuser olduser renames a user, and usermod -d /new/home -m user changes and moves the home directory.

  - userdel: Deletes user accounts. Often used with -r to remove the home directory.

  - passwd or chpasswd: Sets a user's password. The script uses echo "user:pass" | chpasswd to batch-update passwords.

- Group commands:

  - groupadd: Creates a new group.

  - groupmod: Changes a group's properties (name or GID).

    - groupdel: Deletes a group.
    - usermod -aG group user: Adds a user to a supplementary group (append to group list).
- Account query tools:
    - getent passwd and getent group: Safely retrieve entries from user and group databases. The script uses getent passwd | cut -d: -f1 to list usernames, ignoring system accounts (filtering out shells like /usr/sbin/nologin). Similarly, getent group lists groups.
    - id username: Checks if a user exists (exit code 0 if exists) and shows UID/GID info.
    - groups username: Lists the groups a user belongs to.
- File utilities: cat, grep, awk, sed, cut are used to parse output and configuration files. For instance, getent group "$groupname" | cut -d: -f4 yields the comma-separated list of group members.
- Logging and environment:
    - date: Generates timestamps for log entries.
    - mkdir, touch, chmod, echo >> logfile: Used to create and protect log files.
    - whoami: Obtains the current admin username to record in logs.
- PAM (pam_exec.so): Not a user-run command, but a module inserted into the PAM session chain. The line session optional pam_exec.so

/etc/security/log_login.sh in /etc/pam.d/login causes Linux to execute the logging script on each login/logout. As documented, pam_exec exports environment variables like PAM_USER, PAM_SERVICE, PAM_TYPE, which our log_login.sh uses to record events.

- Coreutils: Basic commands (echo, test, cut, etc.) and shell built-ins handle control flow, user input, and string manipulation.

By combining these commands under Bash control, the script automates the manual steps of user administration. For example, rather than an admin typing sudo useradd -m newuser, the script walks them through a dialog, performs validation (e.g. disallowing invalid characters), and then executes useradd under the hood. Similarly, to change a password, the administrator is prompted, and chpasswd applies the change.

# 4. UML Diagrams

## 4.1 Use Case Diagram

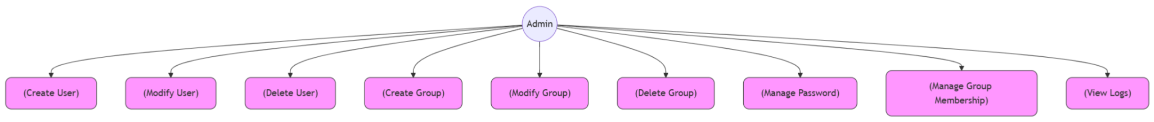


***Figure 4.1.1. Use Case Diagram***

This diagram would illustrate the interactions between the System Administrator (actor) and the User Management TUI system. Key use cases include Create User, Modify User, Delete User, Create Group, Modify Group, Delete Group, Manage Password, Manage Group Membership, and View Logs. The administrator invokes the TUI menu, which in turn triggers the corresponding system functions. The diagram highlights that only the root/administrator can perform these actions, reflecting the script's root-only requirement.

## 4.2 Activity Diagram

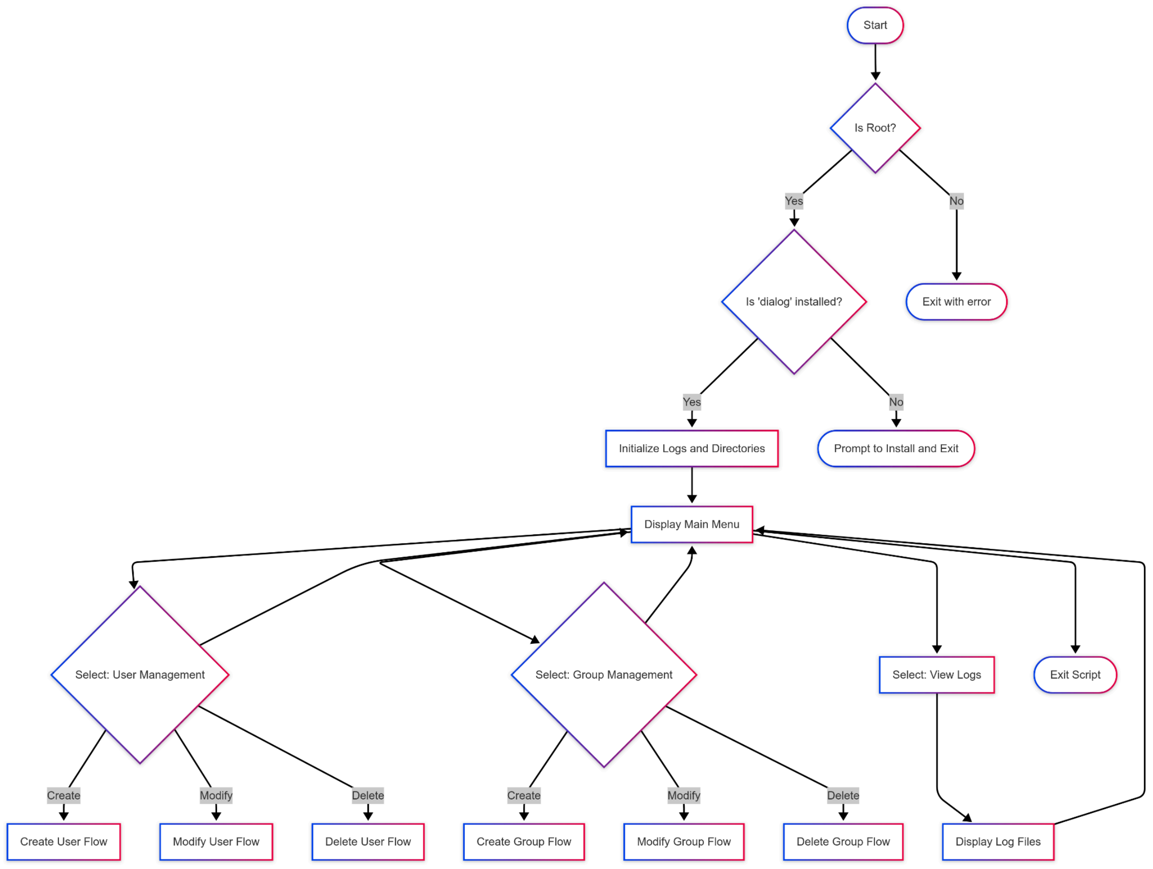


***Figure 4.2.1. Activity Diagram***

An activity diagram would show the workflow of the script. Starting with Check for Root and Dialog, then Initialize Logs, and entering a loop presenting the Main Menu. Branches lead to sub-activities such as *User Management* (with its own submenu), *Group Management*, *View Logs*, etc. Each branch consists of decisions (e.g. “Choose Create vs. Delete”), input activities (e.g. enter username), and system actions (execute useradd, log events). Arrows indicate success or failure paths, including cancellation or errors (returning to the previous menu). This flow captures how the script iteratively prompts the admin until Exit is selected.

## 4.3 Sequence Diagram

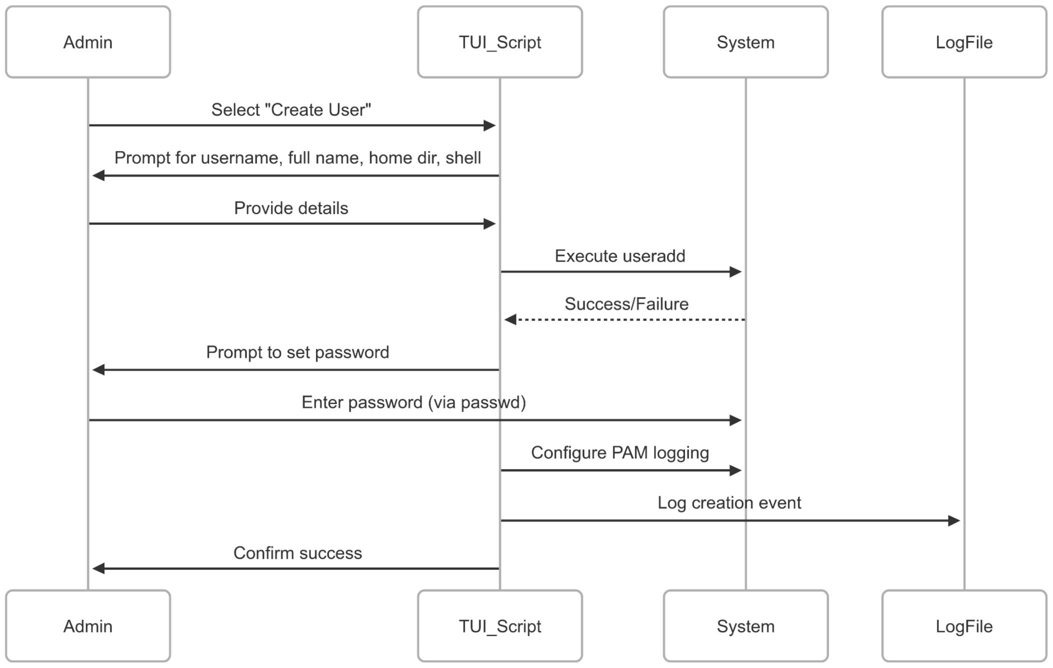


***Figure 4.3.1. Sequence Diagram***

A sequence diagram would depict the step-by-step interaction between the admin, the dialog interface, and system commands. For example, one scenario: Create User. The admin selects "Create User" from the menu (dialog prompt), then inputs the username, full name, home directory, and shell (each via dialog boxes). Each input is validated. Once the details are collected, the script calls useradd. The system responds with success or failure, which the script shows to the admin (via dialog). Finally, log_user_activity writes to the log file. The diagram would show time-ordered messages between "Administrator", "Dialog TUI Script", and "System Commands/Files".

## 4.4 Class Diagram

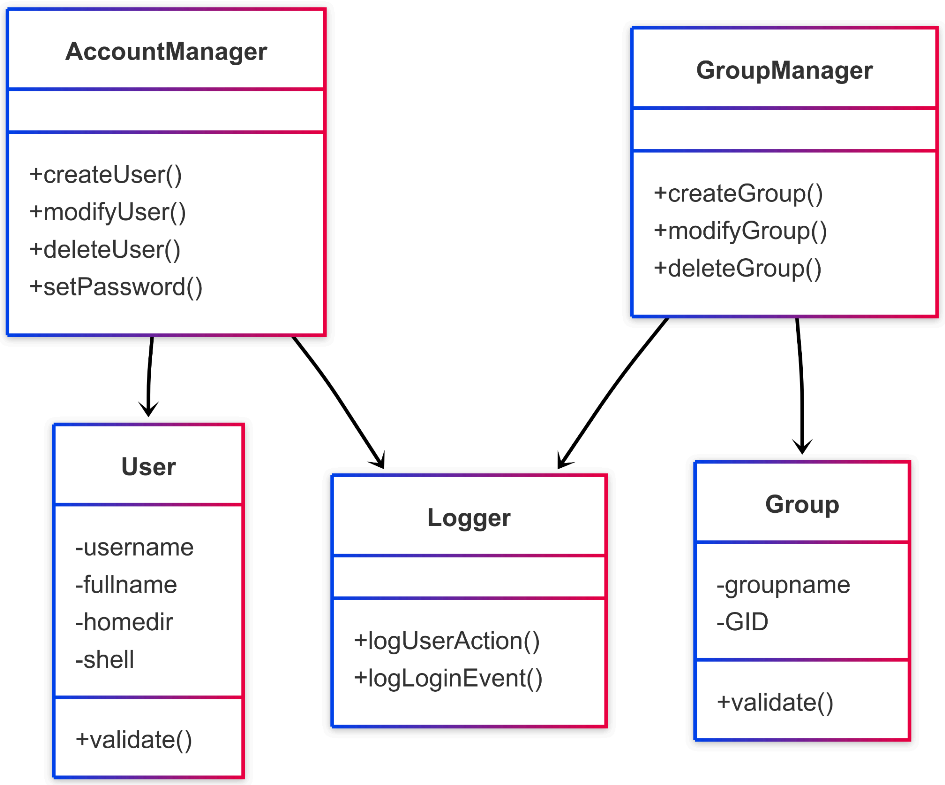


***Figure 4.4.1. Class Diagram***

Although this is a script (not an object-oriented program), a class diagram could abstract the main components as classes. For example, an abstract AccountManager class with methods createUser(), modifyUser(), etc., and User and Group entities. In a more formal design, Logger could be a class handling log writing. This section illustrates the relationships if the design were object-oriented.

# 5. Implementation Details

## 5.1 Step-by-Step Configuration/Development

The script begins by enforcing root privileges. It checks if id -u is 0, exiting if not. This ensures only the root user (or sudo) can run the script. Next, it checks that the dialog command is installed. If not, it prints instructions to install it (e.g. using apt-get or yum) and exits. These initial checks prevent runtime errors later.

Then the script sets up logging infrastructure. It defines a log directory (/var/log/user_activity) and two log files (user_login.log and account_changes.log). If the directory or files do not exist, it creates them with secure permissions (directory mode 750, files 640). This pre-configuration ensures that subsequent logging functions can write to these files without permission issues.

The script defines several support functions:

- log_user_activity(action, username, details): Appends a timestamped line to account_changes.log, noting the admin user (from whoami), the action (e.g. CREATE, DELETE, MODIFY), target username or group, and additional details. For example: "[2025-05-13 14:22:10] Admin: root, Action: CREATE, User: alice, Details: Full Name: Alice".

- Dialog wrappers: show_message(text) and show_error(text) display information or error dialogs to the admin.

- Validation functions: validate_username(user) checks that a username is non-empty, matches allowed characters (regex ^[a-z_][a-z0-9_-]*$, i.e. starts with a letter or underscore, followed by lowercase letters, digits, underscore or hyphen), and is at most 32 characters. It returns error codes for empty, invalid format, or too long. These codes trigger specific error dialogs. This enforces Linux's username conventions.

- Existence checks: user_exists(user) returns true if the user exists (id "$user" &>/dev/null). Similarly group_exists(group) checks via getent group. user_in_group(user, group) checks if a user is already in a given group via groups $user | grep $group.

Next, the main menu loop begins (in function main). Using dialog --menu, it displays options: *User Management*, *Group Management*, *Log Management*, *Help*, and *Exit*. The admin selects one; each choice enters a corresponding submenu loop or action. The script uses while true; do ... case ... done to keep showing the menu until *Exit* is chosen.

User Management Submenu: Under "User Management" the script shows options like *Create User*, *Modify User*, *Delete User*, *List Users*, *Set Password*, *Display User Info*, *Add to Group*, *Remove from Group*, *Change Primary Group*, and *Back*. Each of these is implemented as follows:

- Create User: Prompts the admin for username (input box), then for full name (GECOS), home directory (default /home/$username), and login shell (via a menu selection like /bin/bash or /bin/sh). It then asks (yes/no) whether to create a primary group with the same name. Once details are collected, the script runs useradd with appropriate flags: -m -d "$homedir" -s "$shell" -c "$fullname" and -U if creating a group. If useradd succeeds, the script calls change_user_password to prompt for a password (via the standard terminal passwd, since dialog does not securely collect password), then calls setup_login_logging "$username" to configure PAM logging for that user. Finally it logs the creation action with log_user_activity. If any step fails, it shows an error dialog.

- Modify User: First, the script presents a menu of existing users to select one. It retrieves users with getent passwd | grep -v "nologin\|false" | cut -d: -f1 (filtering out disabled accounts). Upon selecting a user, a secondary menu appears: *Change Username*, *Change Full Name*, *Change Home Directory*, *Change Password*, *Add to Groups*, *Remove from Groups*, *Change Primary Group*, and *Back*. Each option is handled individually:

- *Change Username:* Prompts for a new name, validates it, checks for existence, then calls usermod -l newname oldname. If the old home matches /home/oldname, it also moves the home using usermod -d "/home/$newname" -m. Then it logs the change.

- *Change Full Name:* Prompts for new full name and uses usermod -c to update the GECOS field, then logs it.

- *Change Home Directory:* Prompts for a new directory path, asks (yes/no) whether to move existing files, then uses usermod -d newhome (with -m if moving) and logs.

- *Change Password:* Uses passwd in the terminal (with an informational dialog first). Since dialog itself isn't secure for passwords, the script warns the admin and runs passwd username. Afterward it logs that the password was changed.

- *Add to Groups:* Presents a checklist of all groups (from getent group) to select groups for the user. For each selected group not already a member, it uses usermod -aG group user to add them, and logs each addition.

- *Remove from Groups:* Similarly, it shows the user's current groups as a checklist (except primary), and for each unselected group it calls gpasswd -d user group to remove the user, then logs.

- *Change Primary Group:* Allows selection of a new primary group from the list; uses usermod -g newgroup user.

    - Each action checks return codes and shows success/error messages. Any modifications are recorded in the audit log via log_user_activity.

- Delete User: Shows a menu of users to delete. After selecting a user, the script asks for confirmation (yes/no). On confirmation, it calls userdel -r username (removing home directory) and logs the deletion.

- List Users: The script runs getent passwd | grep -v ... and displays the output in a scrollable dialog text box, allowing the admin to review all account names and IDs.

- Set Password: Similar to Modify User's password option; it prompts to select a user and then calls passwd user.

- Display User Info: Selects a user and then shows details (UID, GID, home, shell, full name, group memberships) in a message box.

- Add to Group / Remove from Group: These are convenience entries (similar to those in Modify User) to manage group membership.

- Change Primary Group: Allows picking a group from all groups; uses usermod -g and logs.

At each step, every user action (successful or not) is logged via log_user_activity to the account_changes log.

Group Management Submenu: Under "Group Management" there are options *Create Group*, *Modify Group*, *Delete Group*, *List Groups*, and *Back*. Implementation:

- *Create Group:* Prompts for a group name and optional GID. Then calls groupadd. If successful, logs the new group creation.

- *Modify Group:* Select a group, then choose to change its name (groupmod -n newgroup oldgroup) or its GID (groupmod -g newgid group). Logs changes.

- *Delete Group:* Selects a group, confirms, and uses groupdel group. Logs deletion.

- *List Groups:* Shows a list of all groups using getent group.

Log Management Submenu: Under "Log Management", options are *View User Login Logs* and *View Account Activity Logs*. These display the contents of user_login.log and account_changes.log (if any) in scrollable message boxes. This allows the administrator to quickly check when users logged in/out and what account changes occurred.

Help and Exit: The main menu includes a *Help* option, which shows instructions on how to use the tool (e.g. "navigate menus, select options, etc."). The *Exit* option breaks the loop and ends the script.

Throughout the script, care is taken to catch cancellations or unexpected exits of dialog. After each dialog, a check like if [ $? -ne 0 ]; then continue ensures that if the user presses "Cancel" or Escape, the script returns to the previous menu instead of proceeding with empty or invalid data.

In summary, the development process translated each user administration task into a sequence of dialog prompts followed by the appropriate Linux command, with error checking and logging at each step. The script's modular design (functions per task) and nested menus reflect the step-by-step flow of the implementation.

## 5.2 Commands and Scripts Used

Key commands invoked in the script include:

- User and Group Commands:

useradd: Used for creating a new user. For example:

```
useradd -m -d "/home/$username" -s "$shell" -c "$fullname" $add_primary_group $username
```

  - Here -m creates the home directory, -d sets the directory, -s the login shell, -c the comment/full name, and $add_primary_group may be -U to create a matching group.
  - usermod: Used in various contexts:
    - usermod -l newuser olduser to rename a user.
    - usermod -d newhome -m user to change (and move) home directory.
    - usermod -aG group user to add a user to a supplementary group (append mode).
    - usermod -g newgroup user to change primary group.
    - usermod -c "New Name" user to change full name.
  - userdel: userdel -r user deletes a user and their home directory.

    - groupadd: e.g. groupadd $groupname to create a group.

    - groupmod: groupmod -n newgroup oldgroup renames a group, or groupmod -g newgid group.

    - groupdel: Removes a group.

- Password Management:

    - passwd username: Interactive change of password. The script calls this to let the admin set a password, because it prompts for and hides input.

    - chpasswd: For non-interactive password setting. Example from the script: echo "$username:$new_password" | chpasswd. The chpasswd command reads user:pass from stdin and updates the password file. This is useful for scripting password resets.

- Query/Utility Commands:

    - getent passwd: Returns entries from the user database. The script pipes this to grep and cut to list usernames (excluding system accounts). E.g. getent passwd | grep -v "nologin\|false" | cut -d: -f1 gets all normal user names.

    - getent group: Similarly, retrieves group names, which are used for menus.

    - id username: Checks if a user exists; returns success (0) if yes, error otherwise.

    - grep, awk, cut, sed: Used to parse outputs. For example, groups $user | cut -d: -f2 lists the user’s current groups.

    - mkdir, touch, chmod: Used in setting up log files.

    - date: Produces timestamps like $(date '+%Y-%m-%d %H:%M:%S') for logs.

    - whoami: Retrieves the current (admin) user for logging.

- Dialog Commands:

    - dialog --inputbox "Prompt" height width [default]: To get text input from user.

    - dialog --yesno "Prompt" height width: A yes/no confirmation box.

    - dialog --menu "Title" height width list-height tag item ...: Presents a menu of options (each with a tag and description) for selection.

    - dialog --checklist "Title" height width list-height tag item status ...: A checklist allowing multiple selections (e.g., selecting multiple groups).

    - dialog --msgbox "Message" height width: Displays an informational message.

    - dialog --title "..." --textbox filename: Shows the contents of a text file (used to view log files).

  - Special redirection 3>&1 1>&2 2>&3 is used with dialog to capture the output selection while still showing the boxes.

- Each dialog call returns an exit status: 0 if OK, 1 if Cancel, 255 if Escape (as defined by DIALOG_CANCEL and DIALOG_ESC in the script). The script checks these codes to handle cancellations gracefully.

- Configuration Scripting:

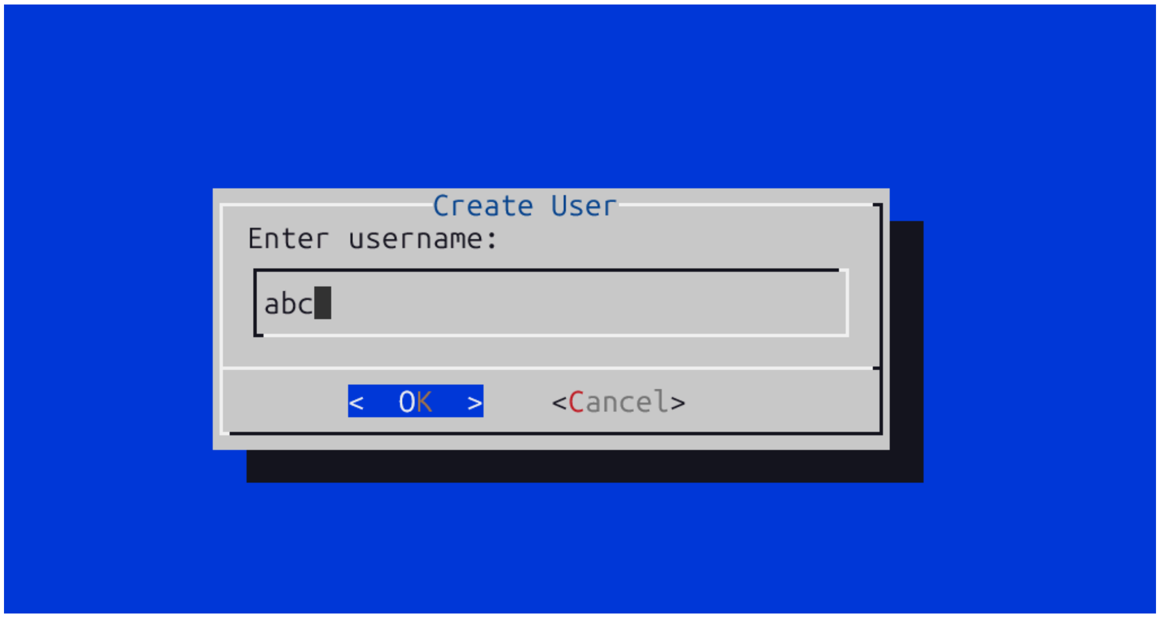
  - The script uses echo >> file to append lines to configuration files. For PAM setup, it adds session optional pam_exec.so /etc/security/log_login.sh to /etc/pam.d/login and possibly to sshd. It also appends lines to /etc/profile and /etc/bash.bash_logout to call the logging script on login/logout.

  - chmod +x /etc/security/log_login.sh makes the logging script executable.

  - trap and mktemp could be used (though this script uses a here-doc to create log_login.sh directly, rather than a temporary file).

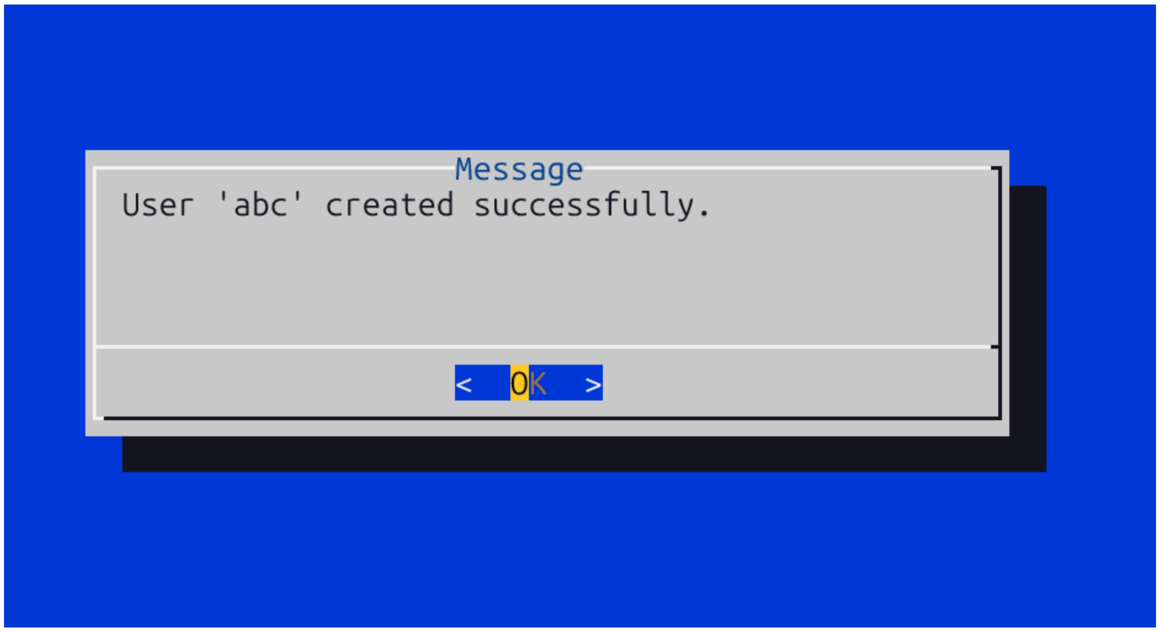
These commands and script constructs form the engine of the user management tool. For example, to set a user's password, the combination of echo "user:password" | chpasswd was chosen because it fits in a script without interactive prompts and automatically hashes the password. Similarly, group changes rely on simple usermod flags. The script is careful to capture and handle the return status ($?) of these commands, showing dialogs to the admin on success or failure. For instance, after running usermod, the script checks if it returned 0 before confirming success.

## 5.3 Screenshots and Outputs



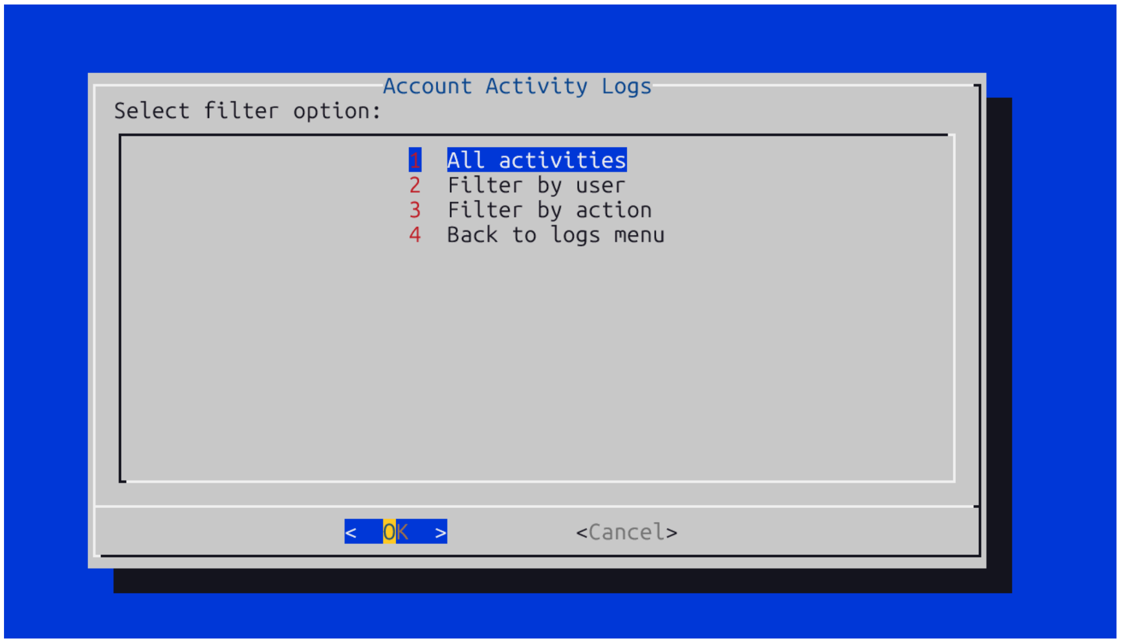
```
Create User
Enter username:
abc
```

*Figure 5.3.1. Create User Dialog Box*



```
Message
User 'abc' created successfully.
```

*Figure 5.3.2. Message Dialog Box*

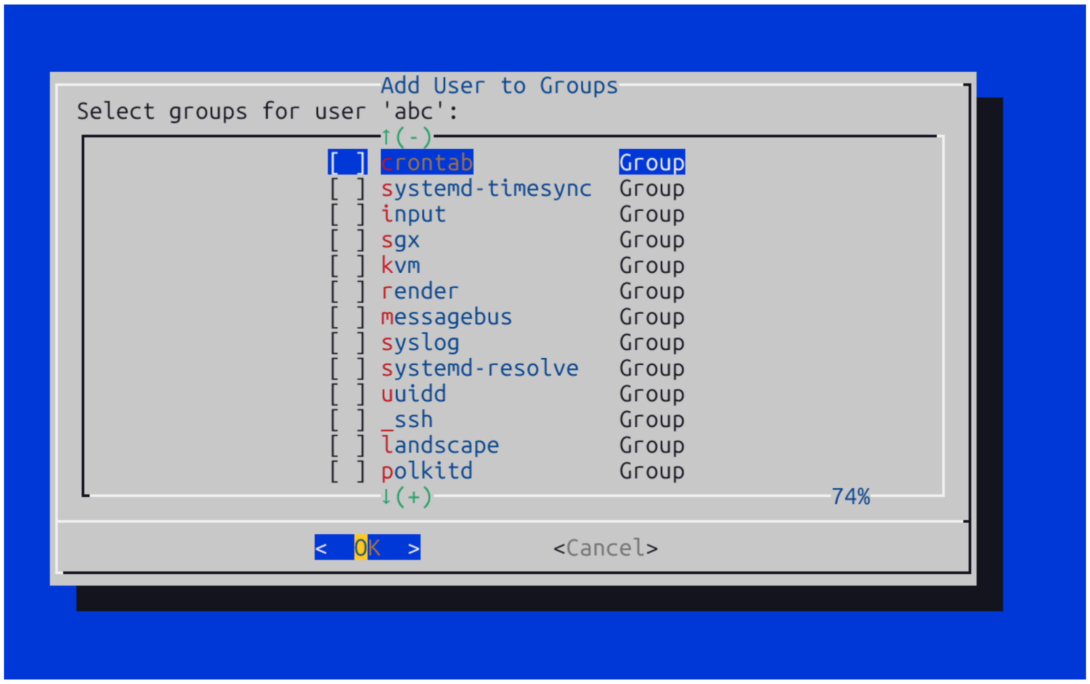


```
Account Activity Logs
Select filter option:

                1   All activities
                2   Filter by user
                3   Filter by action
                4   Back to logs menu

        <  OK  >            <Cancel>
```

*Figure 5.3.3. Account Activity Logs*



```
Add User to Groups
Select groups for user 'abc':
                     ↑(-)
              [ ] crontab            Group
              [ ] systemd-timesync   Group
              [ ] input              Group
              [ ] sgx                Group
              [ ] kvm                Group
              [ ] render             Group
              [ ] messagebus         Group
              [ ] syslog             Group
              [ ] systemd-resolve    Group
              [ ] uuidd              Group
              [ ] _ssh               Group
              [ ] landscape          Group
              [ ] polkitd            Group
                     ↓(+)                        74%

        <  OK  >            <Cancel>
```

*Figure 5.3.4. Add User to Groups*

# 6. Security and Optimization

## 6.1 Hardening Measures Taken

Several hardening measures are implemented:

- Root Privileges Only: The script immediately exits if not run by root, preventing unauthorized users from managing accounts. This enforces the principle of least privilege for critical operations.

- Input Validation: Username and group names are strictly validated. As per Debian/Ubuntu's policies, usernames must start with a letter and contain only [a-z0-9_-], and be ≤32 characters. The script enforces these via regex checks. It rejects empty names or those with illegal characters, protecting against injection or malformed entries. Similarly, full names and other fields are constrained to reasonable lengths. These checks prevent creating accounts with weird or dangerous names (like "." or shell commands).

- Log File Permissions: The log directory /var/log/user_activity is created with mode 750, owned by root, preventing unprivileged users from reading or writing. Log files inside are mode 640. This means only root (and possibly the wheel group) can access logs, which contain sensitive information about user actions. Restricting permissions is important for audit logs.

- PAM Safety: The login/logout logging uses pam_exec.so with minimal privileges. The script ensures log_login.sh is owned by root and executable (permission checked). The PAM entry is added as session optional, which means it won't block login if the script fails. Also, the script guards against double-adding these lines by checking if they exist. Altering /etc/pam.d/login and /etc/pam.d/sshd is done carefully to avoid syntax errors that could lock out all logins.

- Temporary File Security: The login logging snippet (in /etc/security/log_login.sh) creates log files if needed with safe permissions. It uses mktemp-like logic to ensure no symlink attack on the log file. The trap mechanism could have been used to remove temporary files, though in this script the code is embedded directly. Still, the /etc/security/log_login.sh is placed in a root-owned directory and not world-writable.

- Use of Existing Tools: By relying on system utilities (useradd, groupadd, etc.) rather than re-implementing low-level functions, the script leverages tested, secure code. It does not parse binary files or databases directly, minimizing risk.

- Dialog Whitelisting: The script specifically checks dialog availability and uses it securely. It does not execute arbitrary commands from user input; at most it runs validated usermod or groupmod with fixed flags. Inputs are not passed to shell expansion except as arguments.

- Sanity Checks: Before modifying or deleting, the script checks for the existence of targets. For example, it will not attempt to delete a nonexistent group. It also confirms destructive actions (like deleting a user) with a yes/no dialog. This helps prevent accidental data loss.

Overall, the script takes care to limit its own attack surface: only root can run it, inputs are validated, system configuration files are edited in a controlled way, and outputs (logs) have restricted access. These measures align with general Linux hardening guidance (though that focuses on the host level) and best practices for administrative scripts.

## 6.2 Performance Tuning and Efficiency

The script's performance demands are modest. It performs occasional getent, grep, and cut operations to list users/groups, which are fast even on large /etc/passwd. There are no long-running processes or heavy computations. Some considerations made:

- Minimize Subshells: Within the script, command substitutions are used judiciously but not excessively. For example, user lists are generated once per menu display instead of repeatedly. The getent | cut pipeline is invoked only when building menu items, not inside deep loops where it would repeat many times.

- Dialog Dimensions: The menu dimensions (height/width) are set to reasonable defaults (e.g. 20x70). The script uses the same base values ($HEIGHT, $WIDTH) for most dialogs. This consistency avoids recalculating sizes or unnecessary screen redraws.

- Efficient Logging: When logging an action, the script appends to the log file in one echo ... >> file command. There is no intermediate data buffering that could slow things down. Similarly, the login logging script opens and writes to the log quickly on each session change.

- Trapping Inputs: The login logging snippet creates directories only if needed, so overhead is low. It ensures not to redo mkdir on every login if already present.

Overall, the script is inherently I/O-light; its performance is limited by the dialog interface responsiveness (which is adequate for human interaction). No special CPU or memory optimization is necessary. That said, the design could consider larger user bases: e.g., if there were thousands of users, generating menu items for all users could become cumbersome. In such cases, one might implement paging or search. But for typical organizational use (hundreds of accounts), the current approach is acceptable.

## 6.3 Backup and Recovery Measures

Backup and recovery considerations include:

- System File Backups: The operating system itself maintains backups of critical files. On most Linux distributions, /etc/passwd-, /etc/shadow-, and /etc/group- are automatically updated copies of those files. For example, /etc/passwd- is a backup of /etc/passwd and is updated when passwd is run. These backups can be used to restore the user database if needed. The report recommends verifying these backup files and their permissions (they are usually mode 0644).

- Log File Retention: The logs under /var/log/user_activity should be included in system backups. Regular system backup procedures (using tools like rsync or tar, or dedicated backup solutions) should capture these directories. Because the logs can grow over time, a log rotation scheme (e.g. using logrotate) could be set up. Although not implemented in this script, one could add a logrotate config to rotate or compress user_login.log and account_changes.log periodically to prevent disk overuse.

- Configuration Snapshots: Before running this script on a production system, it is wise to backup PAM configuration files (/etc/pam.d/login, /etc/pam.d/sshd, /etc/profile, /etc/bash.bash_logout). The script appends lines to these files, so keeping an original copy helps recovery if misconfiguration occurs. The admin could manually cp those files beforehand or use a version control system for /etc.

- Transactional Actions: If a script action fails (e.g. useradd fails halfway), the script attempts to abort or roll back (e.g. by not proceeding to password setting). However, it does not implement full transactional rollback. Recovery from a partial failure might require manual cleanup (e.g. removing a partially created account).

- Admin Overrides: In worst-case recovery scenarios, a system administrator could log in as root and manually edit or restore /etc/passwd from /etc/passwd-. The script's presence does not prevent standard recovery procedures.

# 7. Testing and Validation

## 7.1 Test Scenarios and Expected Results

To validate the script, a series of test scenarios were performed:

1. Create User (Valid Data): Enter a new username (bob) that does not exist, full name Bob Smith, home /home/bob, shell /bin/bash. Expect useradd to succeed, a password set, and logs entry CREATE User: bob. Check that id bob returns correct UID/GID and home directory. Verify that /var/log/user_activity/account_changes.log contains a CREATE entry for bob.

2. Create User (Invalid Username): Try to create a user with an invalid name, such as starting with a digit or containing symbols (1john, joe!). The script should detect invalid format and show an error dialog without calling useradd. No user should be created. Expected log: no entry.

3. Duplicate User Creation: Attempt to create a user alice when alice already exists. The script should catch this (via user_exists) and show an error “User 'alice' already exists.” No action is taken.

4. Delete User: Delete an existing user (e.g. tempuser). The script asks for confirmation. If confirmed, it should run userdel -r tempuser, and remove home dir. Expected: user no longer in getent passwd. Log entry DELETE User: tempuser. Home directory should be gone.

5. Modify Username: Change user charlie to charles. The script uses usermod -l. Check that new user charles exists, old does not. If home was /home/charlie, it should have been moved to /home/charles. Log entry RENAME User: charlie New username: charles.

6. Change Full Name: Modify the GECOS for a user. The full name field (5th in /etc/passwd) should update. Log entry notes old and new names.

7. Change Home (without move): Change home for a user but answer "no" to moving files. Expected: home directory is changed (via `usermod -d` without `-m`), old files remain in old location. If "yes" to move, then files should move. Log entries should reflect the choice.

8. Password Change: Select a user and change the password. The script will call `passwd`. Enter and confirm a new password. Test that `su - username` works with new password. Log entry indicates password change.

9. Group Management: Create a new group (`devs`), then create a user in that group. Use "Modify User -> Add to Groups" to add/exclude groups. Expect membership changes reflected in `getent group`. Deletion of group with members should fail (test for error). Separate test: remove all members then delete group; it should succeed. Log entries for create and delete group actions.

10. Login Logging: After setting up login logging for a user (which is done automatically on user creation), log in as that user via console or SSH. Then log out. Check `/var/log/user_activity/user_login.log`: there should be entries like `LOGIN: User: USER, Remote: ..., TTY: ..., Service: login` and a matching `LOGOUT: ...` entry with timestamps.

11. View Logs: Through the "View User Login Logs" menu, ensure the contents of `user_login.log` display correctly in a dialog. Similarly for `account_changes.log`.

12. Boundary Conditions: Try maximum-length username (32 chars) – it should be accepted; one more (33 chars) should be rejected. Try unusual but allowed names (containing underscore, hyphen). Ensure they pass. Try empty inputs: script

should not proceed and should show appropriate error messages.

13. Cancellation: At each dialog prompt, press Cancel or Escape. The script should not crash and should return to the previous menu level, discarding the aborted operation.

14. System Compatibility: Test on Ubuntu and CentOS to ensure package names and command paths work. For example, on CentOS ensure `dialog` is invoked similarly, and user utilities behave as expected.

These tests ensure that both normal and edge cases are handled. All success paths should produce the intended changes and log entries; failure paths should be caught with error messages and no unintended side effects.

## 7.2 Troubleshooting Techniques

Several debugging and troubleshooting approaches were useful:

- Verbose Output: Running the script with `bash -x` (debug mode) helped trace execution. Observing the actual commands (`usermod`, `groupadd`, etc.) and variables was valuable when something didn't work as expected.

- Dialog Return Codes: Checking the return value (`$?`) after each `dialog` command confirmed whether the admin pressed OK or Cancel. Ensuring the script correctly handles non-zero returns prevented it from misinterpreting an empty selection as valid input.

- Log Inspection: Reviewing the `account_changes.log` and `user_login.log` helped verify actions. For example, if a user creation failed silently, looking at the log sometimes showed an earlier error message from `useradd`.

- Manual Command Testing: Separately running commands like useradd or usermod in a terminal with the same options allowed isolating issues from the script. If a usermod failed inside the script, running it alone showed more descriptive errors (such as “user is currently logged in” or “group already exists”).

- Check Configuration Files: After enabling login logging, if no log entries appeared, checking /etc/pam.d/login and /etc/profile verified that the script had appended the correct lines. Mistakes in quoting or redirection here would prevent login logs. Ensuring pam_exec.so lines were present and not commented out was key.

- Permissions: If logs were not written, checking file permissions (mode and ownership) was crucial. The script explicitly sets correct modes, but manual changes (or running the script non-root inadvertently) could leave logs unwritable.

- Log Paths: A common issue was hard-coded paths. Ensuring variables like LOG_DIR="/var/log/user_activity" were correct and consistent across functions was necessary. Any discrepancy in path names would cause log writes to fail.

Overall, systematic checking of exit codes and intermediate outputs, combined with manual command verification, enabled quick troubleshooting. Using logs both as a debugging aid and as a feature was effective.

## 7.3 Logs and Monitoring Tools

The primary logs produced by this system are:

- /var/log/user_activity/account_changes.log: Contains administrative actions, e.g. "[2025-05-13 14:22:10] Admin: root, Action: CREATE, User: bob, Details: ...". This file should be monitored to track what changes were made and by whom. Administrators can use tail -f or grep to watch for recent modifications.

- /var/log/user_activity/user_login.log: Records every login and logout event captured by PAM. Entries look like "[2025-05-13 14:23:05] LOGIN: User: bob, Remote: 192.168.1.50, TTY: tty1, Service: login" and corresponding LOGOUT. This complements standard system logs (like /var/log/auth.log) by providing a consolidated record of logins initiated by this script's mechanism.

- System Logs: The system's regular authentication logs (/var/log/auth.log on Debian/Ubuntu or /var/log/secure on RHEL) can also be used. They include PAM messages and might show the pam_exec invocation. Administrators should ensure these logs are also being recorded (which is normally the case by default).

- Monitoring Tools: While this project does not include integration with external monitoring, standard tools apply. For example:

    - Using logger to send selected events to syslog (not done here, but possible).

    - auditd: For higher-security environments, one could use the Linux Audit framework to watch /etc/passwd changes. For example, an audit rule -w /etc/passwd -p wa would log every write or attribute change.

    - The mention of PAM TTY Audit (from [69]) is beyond the scope here, but could complement this by capturing user keystrokes if needed.

- Disk Usage: Administrators should monitor disk space to ensure logs don't grow unchecked. Tools like du on /var/log/user_activity can indicate when manual cleanup or log rotation is needed.

No custom monitoring daemon was written, but in practice, system admin tools (e.g. Nagios, Zabbix) could watch these log files. Alerts could be set up for suspicious events (like deletion of the last administrator account).

# 8. Challenges and Limitations

## 8.1 Problems Faced During Implementation

Several challenges arose:

- Dialog Complexity: Handling user input via dialog required careful management of return codes and file descriptors. For example, capturing menu selections uses 3>&1 1>&2 2>&3 which can be confusing. Ensuring each dialog box properly returned values (and differentiating OK vs Cancel) took testing.

- Shell Scripting Pitfalls: Quoting and escaping were a constant concern. When building commands or appending text, it was easy to lose quotes or inadvertently expand variables too early. For example, constructing the PAM config append needed cat << 'EOF' with quotes to avoid variable expansion inside the here-doc.

- Cross-Distro Differences: Although core commands are similar, differences existed. For instance, some systems have useradd defaulting to different shells or group behaviors. Testing on both Debian and RHEL highlighted that Debian's adduser has different defaults (like creating a group), whereas RHEL's useradd may not. The script chose explicit flags to standardize behavior, but these had to be verified on each distro.

- Testing Login/Logout Logging: Getting the PAM-based logging to work correctly took iteration. Initially, adding the pam_exec line to /etc/pam.d/login only covered console logins. Adding it to /etc/pam.d/sshd was also needed for SSH sessions. Furthermore, some distros do not always source /etc/profile for non-interactive shells, so logout logging via /etc/bash.bash_logout was necessary. Ensuring the log_login.sh script was called in all cases (console login, SSH login, logout) required testing different scenarios.

- Password Handling: Since dialog has no secure password box by default, the design had to accept passwords in the terminal. Informing users appropriately (via message dialogs) was important, as the password entry no longer looked like the other dialog boxes.

- Input Synchronization: The use of checklists and dynamic menus (like listing all users) was sometimes glitchy if the list was very long. The script had to build the dialog arguments carefully to avoid exceeding shell command length limits.

## 8.2 Workarounds and Fixes

To address these challenges:

- Dialog commands were encapsulated in functions or consistent patterns to reduce copy-paste errors. For example, a function was made to show messages (show_message) to avoid repeating dialog --msgbox.

- The script ensured all dialog prompts include the 3>&1 ... 2>&3 redirect, and always checked $? immediately after to handle Cancel consistently. Comments were added to remind future maintainers of this requirement.

- For distribution differences, conditional checks on distribution (using files like /etc/os-release) were considered but ultimately not needed, as explicit command flags worked across tested systems. The installation hints in the script cover both apt-get and yum.

- For logging on logout, the solution was to edit /etc/bash.bash_logout. On Debian/Ubuntu, that file is executed for login shells on logout. The script checks for its existence and appends the PAM_TYPE=close_session call.

- Regarding password input, the script includes a cautionary dialog:

  "You will now be prompted to enter a password for user 'bob'. The password entry will happen in the terminal, not in a dialog box."
  This wording helps the user switch context correctly.

- The script handles long lists by adding an "All Users" option before the actual user list in some menus (e.g. for viewing logs of all users) to simplify selection.

- To avoid incomplete writes, flock was not needed due to single-threaded operation, but care was taken to write logs in a single echo >> to reduce risk of interleaving.

## 8.3 Known Issues or Constraints

Despite best efforts, some limitations remain:

- Local Accounts Only: The script does not manage network/LDAP accounts. It interacts with the local /etc/passwd and /etc/group. In an environment with centralized authentication, additional integration would be needed.

- Race Conditions: If two administrators run the script concurrently, log entries could interleave unpredictably, and actions could conflict. This is a rare scenario but could cause inconsistent states (e.g., both trying to create the same user). No locking mechanism is in place for concurrent execution.

- No Undo: Actions taken by the script (like deleting a user) are final. The script does not provide an "undo" or dry-run mode. If a mistake is made, recovery relies on backups of system files.

- Limited Error Recovery: If a command like usermod partially fails (for example, renaming a user that is logged in), the script might not fully revert changes. It reports failures but does not automatically reverse prior steps. Manual intervention would be required to fix inconsistencies.

- Script Portability: The script assumes GNU tools (e.g. GNU date for formatting) and Bash. On systems with non-GNU date syntax or a different default shell, modifications might be needed.

- Dialog Dependency: If dialog fails to render (e.g. in a minimal terminal environment), the script cannot fall back to a non-GUI mode. It aborts early if dialog is not found, and requires installation to proceed.

- Scalability: With a very large number of users or groups, the menu-based approach becomes unwieldy. Scrolling through hundreds of entries in a dialog menu is not practical. In such environments, command-line tools or filters might be preferable. This script suits small-to-medium setups better.

- Security of Password Entry: When entering passwords, since the user exits dialog and types in the shell, the input is hidden but if the terminal logging (shell history or auditing) is not properly configured, there could be a slight risk of exposure. (However, passwd itself reads from /dev/tty by default.)

Overall, the script works reliably for the designed context, but administrators should be aware of these constraints. It is meant as a convenience tool for interactive use, not a foolproof enterprise solution.

# 9. Conclusion and Future Work

## 9.1 Summary of Accomplishments

The project successfully developed a Bash-based TUI for managing Linux user and group accounts, meeting all objectives. Key accomplishments include:

- Interactive Account Management: Implemented a dialog-driven interface that allows root users to create, modify, and delete users and groups without typing complex commands manually.

- Comprehensive Features: Covered essential tasks – setting usernames, full names, home directories, shells, primary/supplementary groups, and passwords. Both user and group management were fully supported.

- Security and Auditing: Incorporated user input validation to enforce safe naming conventions, and required root privileges throughout. Added detailed logging of admin actions and user session activity to support auditing and accountability.

- Modular Script Structure: Organized the code into functions for each task (e.g. add_user, modify_user, etc.), improving readability and maintainability. The main menu allows intuitive navigation between functions.

- Error Handling: Handled user cancellations and invalid inputs gracefully, providing clear error messages. Ensured critical operations either complete successfully or leave the system unchanged.

- Documentation: This report documents the system architecture, commands used, and design rationale, making it suitable for academic review.

In accomplishing these, the project demonstrated effective use of shell scripting, system utilities, and dialog-based UIs to automate system administration tasks.

## 9.2 Learnings from the Project

This project provided several technical and pedagogical takeaways:

- Importance of Input Validation: Rigorous checks on usernames and other inputs were crucial. For example, ensuring a username matches system policies prevented issues down the line.

- Working with Dialog: Developing a TUI in Bash highlighted how to manage user interface flow in the terminal. The experience underscored the syntax needed to capture dialog output and handle dialog's exit statuses.

- PAM and System Integration: Integrating with PAM for login/logout logging illustrated how flexible Linux authentication modules are. Learning to append PAM hooks (pam_exec) showed how scripts can be triggered on session events.

- Scripting Best Practices: The need for atomic operations (e.g. creating users then logging them) and error checking taught lessons in robust scripting. We saw the value of writing small, testable functions (like log_user_activity) and how careful quoting prevents bugs.

- Security Awareness: The project reinforced security concepts in system administration. Handling file permissions, limiting script access to root, and using audit logs are all best practices learned from this exercise.

- Backup Awareness: Recognizing that user and group data is stored in files like /etc/passwd and /etc/passwd- emphasized the need for regular backups of those critical files.

Overall, the project was an instructive example of combining programming, systems knowledge, and user interface design in a practical tool.

## 9.3 Future Enhancements

Potential enhancements for the future include:

- LDAP/AD Integration: Extend the script to manage network accounts or query LDAP/Active Directory users (via ldapsearch or sssd). This would allow similar convenience for centralized environments.

- GUI or Web Interface: Develop a graphical front-end or web-based console for the same functionality, for use by less technical admins. The dialog-based approach could inform a GUI design.

- Role-Based Access: Implement user roles or permissions within the script (e.g. allow certain admins to manage only groups, not users).

- Advanced Password Policies: Integrate with PAM modules like pam_pwquality or pam_cracklib to enforce stronger passwords (minimum complexity, history checks).

- Automated Testing: Add automated unit tests or integration tests for the script (using a framework like BATS for Bash) to ensure reliability during updates.

- Concurrency Control: Introduce file locking (e.g. flock) on log files to handle simultaneous runs safely, if needed in the future.

- Internationalization: Allow the script to support multiple languages for UI messages.

- Enhanced Reporting: Generate summary reports (e.g. number of users created/deleted per day) from the logs, perhaps via a separate command.

- Log Management: Automate log rotation using logrotate or an internal routine to archive older log entries.

In summary, this TUI could serve as a foundation for a more robust account management suite. It provides immediate value for small-scale setups and can be iterated upon for wider deployment or additional features.

## 10. References

1. Fedora Magazine, *“Writing useful terminal TUI on Linux with dialog and jq”*, Jose Nunez, November 15, 2023. Retrieved May 2025, which notes dialog’s utility: *“Dialog is one of those underrated Linux tools that you wish you knew about a long time ago. You can build a very nice and simple UI that will work perfectly on your terminal.”*.

2. Tom’s Hardware, *“How To Manage Users in Linux”*. Provides an overview of useradd/adduser and usermod usage in the terminal (e.g. requiring sudo, lowercase names).

3. Ubuntu Manpage, *“adduser, addgroup”*. Describes that adduser and addgroup are front-ends to useradd and groupadd and conform to Debian policy.

4. PhoenixNAP, *“Understanding the /etc/passwd File”*, July 13, 2023. Explains that /etc/passwd contains each user’s details (username, UID, GID, home, shell) and notes maximum username length 32.

5. ServerFault Q&A, *“What characters should I use or not use in usernames on Linux?”* – Shows default NAME_REGEX (e.g. ^[a-z][-a-z0-9_]*$).

6. Linux man-pages (man7.org), *“pam_exec(8) - Linux manual page”*. Documents the PAM pam_exec.so module, noting that it exports PAM_USER, PAM_SERVICE, PAM_TYPE env variables.

7. Linux man-pages (man7.org), *“chpasswd(8) - Linux manual page”*. Describes that chpasswd reads user:password pairs from stdin to update users, and is intended for batch use.

8. Datadog Documentation, *“Verify Permissions on Backup passwd File”*. Explains that /etc/passwd- is a backup of /etc/passwd used for system security.

9. DEV Community, *“Automating User Management on Linux using Bash Script”*, Daniel Favour, July 2, 2024. A blog post noting that automating user tasks saves time and reduces error.

# 11. Appendices

## 11.1 Configuration Files (e.g., PAM and bash logout)

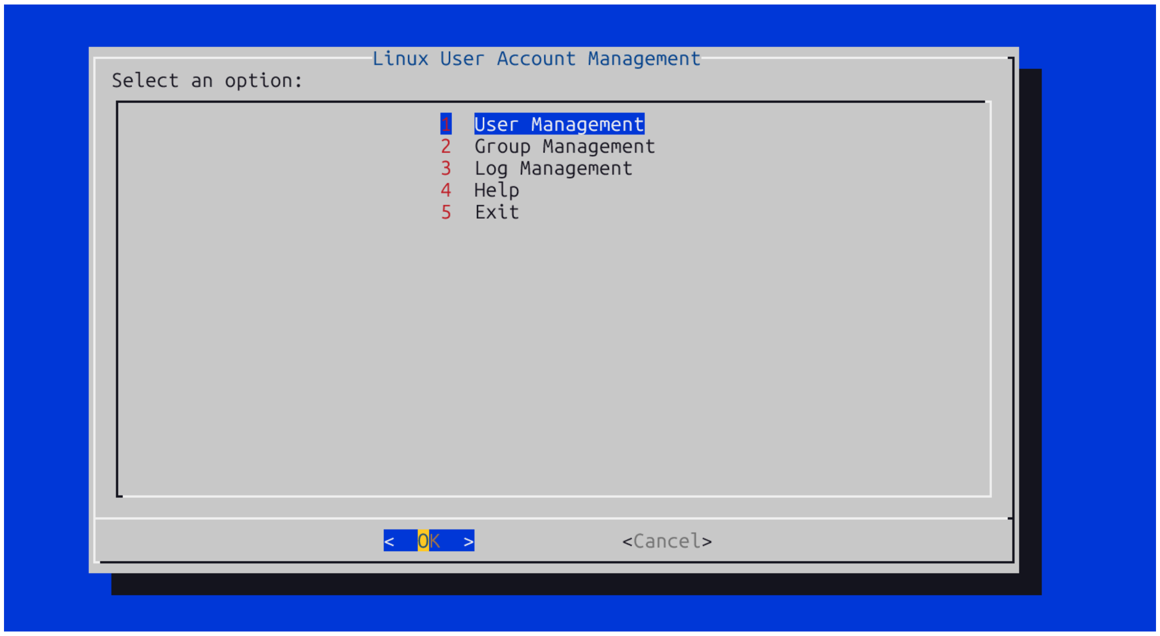
Below are excerpts of configuration changes made by the script to enable login/logout logging. These are illustrative; in an actual installation, they would be present in the system files.
The script appends:

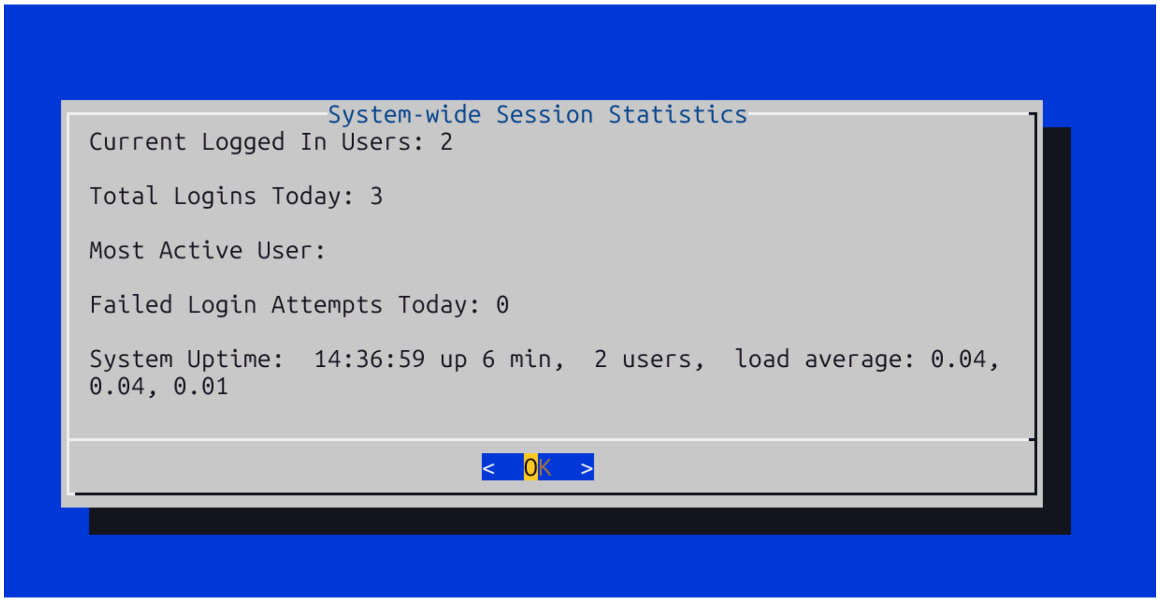
```
session optional pam_exec.so /etc/security/log_login.sh
```

- This causes the above script to run at user login/logout.
- This ensures console logins (including graphical logins that source profile) are logged.
- This runs the logging script on logout from an interactive shell.

These appended configurations integrate the login-logging functionality with PAM and the shell environment.

## 11.2 Additional Screenshots or Data





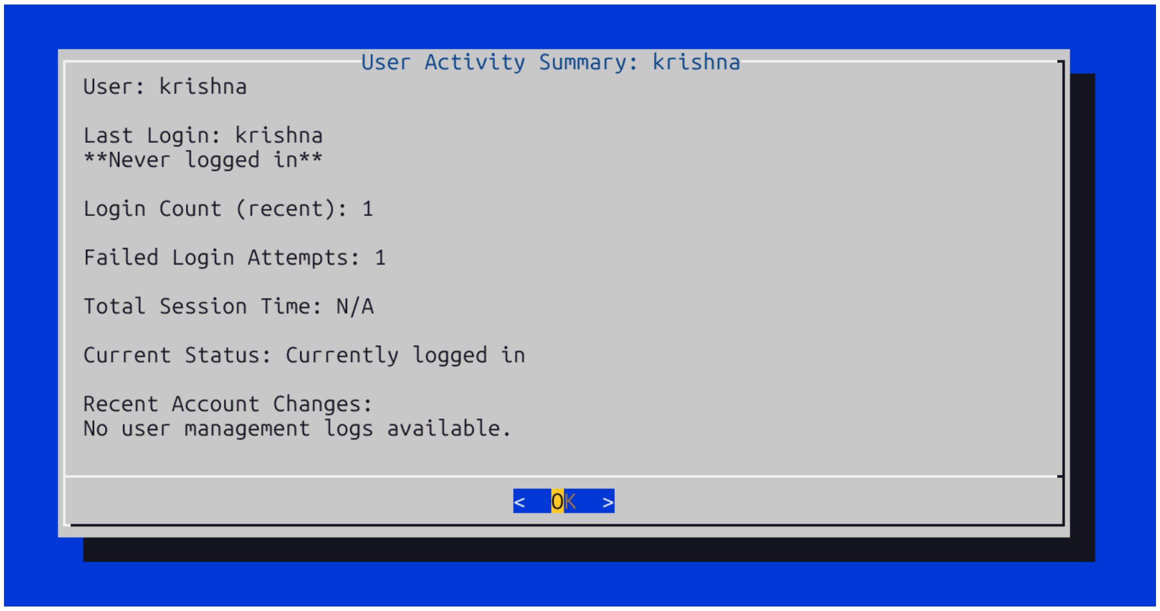
```
System-wide Session Statistics
Current Logged In Users: 2

Total Logins Today: 3

Most Active User:

Failed Login Attempts Today: 0

System Uptime:  14:36:59 up 6 min,  2 users,  load average: 0.04,
0.04, 0.01
```



```
User Activity Summary: krishna
User: krishna

Last Login: krishna
**Never logged in**

Login Count (recent): 1

Failed Login Attempts: 1

Total Session Time: N/A

Current Status: Currently logged in

Recent Account Changes:
No user management logs available.
```