Bash Script Project Documentation

Project Title: User Management Script

Introduction:

The **User Management Script** is designed to assist system administrators in managing user accounts on a Linux-based system. The script allows the creation, modification, deletion of users, assignment of permissions, group management, and more. It also offers a password policy management feature, user activity reports, and logging of all actions performed. This simplifies the task of managing users and their associated configurations.

Author:

- Rhosebrian Jamisola, Clarences Japinan, Jorge Magno, January Iverson Libutan, Arvin Colonia **Version:**
- 2.0

Project Overview:

The **User Management Script** facilitates the management of system users and related configurations through an interactive command-line interface. Administrators can use the script to add or delete users, modify their groups, set permissions for files or directories, generate user activity reports, and enforce password expiration policies. The script automates common administrative tasks and logs all activities to provide traceability.

Features:

- **Add User**: Adds a new user to the system, with validation for password strength.
- **Delete User**: Deletes an existing user along with their files (home directory, mail spool).
    - **Modify User**: Allows changing a user's username or group memberships.
- **Assign Permissions**: Assigns specific file or directory permissions to a user.
- **Assign Groups**: Adds a user to one or more groups.
- **Generate User Activity Report**: Generates a report on user activities, such as last login times and shell information.

- **Set Password Expiration Policy**: Enforces or removes a password expiration policy for users.
- **Logging**: Every action performed by the script is logged with a timestamp in `/var/log/user_management.log`.

Usage:

Command-line:

To run the script, make sure it's executable first. If it is not, use the following command: `chmod +x user_management.sh`

Once the script is executable, run the script by entering the following: `Sudo bash ./user_management.sh`

Menu Options:

1. Add User

- **Purpose**: This option allows the administrator to add a new user to the system.
- **Usage**:
  - After selecting option 1, you will be prompted to enter the following information:
    - **Username**: The username for the new user.
    - **Password**: You will be asked to enter a password for the user. The password must meet security requirements (at least 8 characters, including uppercase, lowercase, digits, and special characters).
    - **Full Name (Optional)**: You can optionally specify the user's full name, or you can press Enter to skip this.
  - Once the information is entered and verified, the script will create the user and log the action.
- **Example**:

```
--- User Management Script ---
1. Add User
2. Delete User
3. Modify User
4. Assign Permissions
5. Assign Groups
6. Generate User Activity Report
7. Set Password Expiration Policy
8. Exit
Choose an option: 1
Enter username: user02
Enter password for user02:
Name (Enter to skip): Clarences Japinan
User user02 added successfully.
```

Select `1` to add a user and follow the prompts to input the username, password, and full name.

2. Delete User

- **Purpose**: This option allows the administrator to delete an existing user and their associated files (e.g., home directory, mail spool). • **Usage**:
  - After selecting option 2, the administrator will be prompted to enter the **username** of the user to delete.
  - The script checks if the user exists and will proceed to delete the user and their files.
- **Example**:

```
--- User Management Script ---
1. Add User
2. Delete User
3. Modify User
4. Assign Permissions
5. Assign Groups
6. Generate User Activity Report
7. Set Password Expiration Policy
8. Exit
Choose an option: 2
Enter username to delete: user02
User user02 deleted successfully.
```

Select `2` to delete a user, then input the username to be deleted.

3. Modify User

- **Purpose**: Allows modification of a user's attributes, such as username and group membership.
- **Usage**:
  - After selecting option 3, the administrator will be prompted to input the **current username**.
  - The script will then prompt you with two modification options:
    - **1**: Change the user's groups.
    - **2**: Change the user's username.

- Depending on your choice, you will be asked to enter the necessary details for the modification (e.g., new username or new groups).

- **Example**:

```
--- User Management Script ---
1. Add User
2. Delete User
3. Modify User
4. Assign Permissions
5. Assign Groups
6. Generate User Activity Report
7. Set Password Expiration Policy
8. Exit
Choose an option: 3
Enter current username to modify: user02
What would you like to modify?
1. Change groups
2. Change username
Choose an option (1 or 2): 2
Enter the new username: user03
User user02 renamed to user03 successfully.
```

Select `3` to modify a user, and follow the prompts to change the username or group membership.

4. Assign Permissions

- **Purpose**: Assign specific file/directory permissions to a user.
- **Usage**:
  - After selecting option 4, the administrator will be asked to provide:
    - **Directory path**: The directory or file whose permissions you wish to modify.
    - **Username**: The user to whom the permissions will be assigned.
    - **Permissions**: The permissions you want to assign (e.g., `755` for full access to the user and read/execute for others).
  - The script will apply the ownership and permissions to the specified directory and log the action.
- **Example**:

```
--- User Management Script ---
1. Add User
2. Delete User
3. Modify User
4. Assign Permissions
5. Assign Groups
6. Generate User Activity Report
7. Set Password Expiration Policy
8. Exit
Choose an option: 4
Enter directory path: /home/Jamisola/FinalProject
Enter username: user03
Enter permissions: 755
Permissions for /home/Jamisola/FinalProject set successfully.
```

Select `4` to assign permissions, then input the directory path, username, and the desired permissions.

5. Assign Groups

- **Purpose**: Add a user to a specified group.
- **Usage**:
  - After selecting option 5, the administrator will be prompted to enter:
    - **Username**: The user to add to a group.
    - **Group**: The group to which the user will be added.
  - If the group exists, the script will add the user to it and log the action.
- **Example**:

```
--- User Management Script ---
1. Add User
2. Delete User
3. Modify User
4. Assign Permissions
5. Assign Groups
6. Generate User Activity Report
7. Set Password Expiration Policy
8. Exit
Choose an option: 5
Enter username to assign group: user03
Enter the group to assign to user03: user
User user03 successfully added to group user.
```

Select `5` to assign a user to a group, and input the username and group name.

6. Generate User Activity Report

- **Purpose**: Generates a detailed report of all users and their activity, including their last login times.
- **Usage**:
  - After selecting option 6, the script will generate the user activity report, which will be saved to a file named `report.txt`.
- **Example**:

```
--- User Management Script ---
1. Add User
2. Delete User
3. Modify User
4. Assign Permissions
5. Assign Groups
6. Generate User Activity Report
7. Set Password Expiration Policy
8. Exit
Choose an option: 6
Report generated: report.txt
```

Select `6` to generate the user activity report. The report will be saved as `report.txt` in the current directory.

7. Set Password Expiration Policy

- **Purpose**: Set or remove the password expiration policy for a user.
- **Usage**:
  - After selecting option 7, the administrator will be presented with two options:

- **1**: Set a password expiration period in days (ex. 30 days).
- **2**: Remove password expiration.

  - After selecting one of the options, you will be prompted to enter the **username** and, if applicable, the number of days for expiration.
  - The script will apply the policy and log the action.

- **Example**:

```
--- User Management Script ---
1. Add User
2. Delete User
3. Modify User
4. Assign Permissions
5. Assign Groups
6. Generate User Activity Report
7. Set Password Expiration Policy
8. Exit
Choose an option: 7
Enter username: user03
Password Policy Options:
1. Set expiration days(ex.30)
2. Remove password expiration
Choose an option: 1
Enter days until password expiration: 20
Password expiration policy set for 'user03'.
```

```
--- User Management Script ---
1. Add User
2. Delete User
3. Modify User
4. Assign Permissions
5. Assign Groups
6. Generate User Activity Report
7. Set Password Expiration Policy
8. Exit
Choose an option: 7
Enter username: user03
Password Policy Options:
1. Set expiration days(ex.30)
2. Remove password expiration
Choose an option: 2
'user03' no longer has a password expiration policy.
```

Select `7` to set or remove a password expiration policy. Follow the prompts to input the username and expiration details.

8. Exit

- **Purpose**: Exits the script.
- **Usage**: Simply select option 8 to exit the script. The script will terminate, and the user will be returned to the command prompt.

Output:

- **Logs**: All actions are logged in `/var/log/user_management.log`.

```
Jamisola@Jamisola:~/User_Management$ cat /var/log/user_management.log
2024-12-14 16:01:53 - Added user: user02
2024-12-14 16:03:33 - Deleted user: user02
2024-12-14 16:13:28 - Added user: user02
2024-12-14 16:14:09 - Renamed user user02 to user03
2024-12-14 16:17:41 - Set permissions for /home/Jamisola/FinalProject to 755 for user user03.
2024-12-14 16:20:06 - Assigned user user03 to group user
2024-12-14 16:22:28 - Set password expiration for 'user03' to '20' days.
2024-12-14 16:23:19 - Removed password expiration for 'user03'.
Jamisola@Jamisola:~/User_Management$
```

- **Reports**: The `generate_report` function creates a file named `report.txt` that contains the user activity report.

```
Jamisola@Jamisola:~/User_Management$ cat report.txt
User Activity Report:
-------------------------------------------------------
Username         Shell           Last Login Time
-------------------------------------------------------
Jamisola        /bin/bash       Sat Dec 14 15:50:48 +0000
user03          /bin/bash       **Never logged in**
Jamisola@Jamisola:~/User_Management$
```

Requirements:

1. **Operating System**: A Linux-based system with `bash` shell.
   (tested on Ubuntu 24.04.1 LTS).
2. **Package Installed:** sudo apt update sudo apt install sudo adduser util-linux

- Sudo privileges for user management actions (e.g., adding/removing users).
- `/var/log/` directory should be writable for logging user actions.

Script Explanation:

Full Script:

```
  GNU nano 7.2
user_management.sh
#!/bin/bash


LOGFILE="/var/log/user_management.log"


# Function to log actions

log_action() {

    echo "$(date +'%Y-%m-%d %H:%M:%S') - $1" >> "$LOGFILE"

}


# Function to check password strength

check_password_strength() {

    local PASSWORD=$1

    if [[ ${#PASSWORD} -lt 8 || ! $PASSWORD =~ [A-Z] || ! $PASSWORD =~ [a-z] || !
$PASSWORD =~ [0-9] || ! $PASSWORD =~ [^a-zA-Z0-9] ]]; then

        echo "Weak password. Must be at least 8 characters long and include upper case, lower case,
digit, and special character."

        return 1

    fi

    return 0

}


# Function to add a user

add_user() {

    read -p "Enter username: " USERNAME
```

```
    # Check if the user already exists
    if id "$USERNAME" &>/dev/null; then
        echo "Error: User '$USERNAME' already exists."
        return
    fi

    # Prompt for and validate password strength
    while true; do
        read -s -p "Enter password for $USERNAME: " PASSWORD
        echo
        check_password_strength "$PASSWORD"
        if [[ $? -eq 0 ]]; then break; else echo "Please try again with a stronger password."; fi
    done

    # Optional full name
    read -p "Name (Enter to skip): " FULLNAME

    # Create user with adduser or fallback to useradd
    if command -v adduser &>/dev/null; then
        echo -e "$PASSWORD\n$PASSWORD" | sudo adduser "$USERNAME" --gecos
"$FULLNAME" --disabled-password > /dev/null 2>&1
    else
        sudo useradd -m -c "$FULLNAME" -s /bin/bash "$USERNAME" > /dev/null 2>&1
        echo "$USERNAME:$PASSWORD" | sudo chpasswd > /dev/null 2>&1
    fi

    # Check for success
```

```
    if id "$USERNAME" &>/dev/null; then
        # Create mail spool for the new user
        sudo touch /var/mail/"$USERNAME"
        sudo chown "$USERNAME":"$USERNAME" /var/mail/"$USERNAME"
        sudo chmod 660 /var/mail/"$USERNAME"

        log_action "Added user: $USERNAME"
        echo "User $USERNAME added successfully."
    else
        echo "Error: Failed to create user '$USERNAME'."
    fi
}
# Function to delete a user
delete_user() {
    read -p "Enter username to delete: " USERNAME
    if id "$USERNAME" &>/dev/null; then
        sudo userdel -r "$USERNAME"
        log_action "Deleted user: $USERNAME"
        echo "User $USERNAME deleted successfully."
    else
        echo "Error: User '$USERNAME' does not exist."
    fi
}

# Function to modify a user
modify_user() {
    read -p "Enter current username to modify: " USERNAME
```

```
if ! id "$USERNAME" &>/dev/null; then
    echo "Error: User '$USERNAME' does not exist."
    return
fi

echo "What would you like to modify?"
echo "1. Change groups"
echo "2. Change username"

read -p "Choose an option (1 or 2): " MODIFY_OPTION

case $MODIFY_OPTION in
    1)
        read -p "Enter new groups: " GRPS
        sudo usermod -G "$GRPS" "$USERNAME"
        log_action "Modified groups for $USERNAME to $GRPS"
        echo "User $USERNAME groups modified successfully."
        ;;
    2)
        read -p "Enter the new username: " NEW_USERNAME

        if id -u "$NEW_USERNAME" &>/dev/null; then
            echo "Error: The username '$NEW_USERNAME' already exists."
            return
        fi

        sudo usermod -l "$NEW_USERNAME" "$USERNAME"
        log_action "Renamed user $USERNAME to $NEW_USERNAME"
```

```
            echo "User $USERNAME renamed to $NEW_USERNAME successfully."
            ;;
        *)
            echo "Invalid option. Please try again."
            ;;
    esac
}

# Function to assign user to a group
assign_group() {
    read -p "Enter username to assign group: " USERNAME
    if ! id "$USERNAME" &>/dev/null; then
        echo "Error: User '$USERNAME' does not exist."
        return
    fi

    read -p "Enter the group to assign to $USERNAME: " GROUP
    if getent group "$GROUP" > /dev/null 2>&1; then
        sudo usermod -aG "$GROUP" "$USERNAME"
        log_action "Assigned user $USERNAME to group $GROUP"
        echo "User $USERNAME successfully added to group $GROUP."
    else
        echo "Error: Group '$GROUP' does not exist."
    fi
}

# Function to assign directory permissions
assign_permissions() {
```

```
    read -p "Enter directory path: " DIR
    read -p "Enter username: " USERNAME
    read -p "Enter permissions: " PERMISSIONS

    # Check if the user exists
    if ! id "$USERNAME" &>/dev/null; then
        echo "Error: User '$USERNAME' does not exist."
        return
    fi

    # Check if the directory exists
    if [ ! -d "$DIR" ]; then
        echo "Error: Directory '$DIR' does not exist."
        return
    fi

    # Set ownership and permissions
    if sudo chown "$USERNAME" "$DIR" && sudo chmod "$PERMISSIONS" "$DIR"; then
        log_action "Set permissions for $DIR to $PERMISSIONS for user $USERNAME."
        echo "Permissions for $DIR set successfully."
    else
        log_action "Failed to set permissions for $DIR with $PERMISSIONS for user
$USERNAME."
        echo "Error: Failed to set permissions for $DIR."
    fi
}

# Function to generate user activity report
```

```
generate_report() {
    # Output file for the report
    echo "User Activity Report:" > report.txt
    echo "-------------------------------------------------------------" >> report.txt
    echo "Username        Shell          Last Login Time" >> report.txt
    echo "-------------------------------------------------------------" >> report.txt

    # Get a list of users with their shells and UIDs from /etc/passwd, excluding system users and nologin users
    awk -F: '{if ($3 >= 1000 && $7 != "/usr/sbin/nologin" && $7 != "/bin/false") print $1, $7}' /etc/passwd | while read -r user shell; do
        # Get the last login information using lastlog
        last_login=$(lastlog -u "$user")

        # Check if the lastlog entry shows "Never logged in"
        if echo "$last_login" | grep -q "Never logged in"; then
            last_time="**Never logged in**"
        else
            # Extract the last login time and avoid printing "Latest"
            last_time=$(echo "$last_login" | awk '{if ($4 != "Latest") print $4, $5, $6, $7, $8}')
        fi

        # Format and write the output to the report
        printf "%-15s %-15s %-40s\n" "$user" "$shell" "$last_time" >> report.txt
    done

    # Confirmation message
    echo "Report generated: report.txt"
}
```

```
# Function to set password expiration policy
set_password_policy() {
  read -p "Enter username: " USERNAME

  # Check if the user exists.
  if ! id "$USERNAME" &>/dev/null; then
     echo "Error: User '$USERNAME' does not exist."
     return
  fi

  echo "Password Policy Options:"
  echo "1. Set expiration days(ex.30)"
  echo "2. Remove password expiration"

  read -p "Choose an option: " OPTION

  case $OPTION in
     1)
        read -p "Enter days until password expiration: " DAYS
        sudo chage -M "$DAYS" "$USERNAME"
        log_action "Set password expiration for '$USERNAME' to '$DAYS' days."
        echo "Password expiration policy set for '$USERNAME'."
        ;;
     2)
        sudo chage -M -1 "$USERNAME"
        log_action "Removed password expiration for '$USERNAME'."
        echo "'$USERNAME' no longer has a password expiration policy."
        ;;
```

```
        *)
            echo "Invalid option. Please try again."
            ;;
    esac
}

# Main menu loop
while true; do
    echo ""
    echo "--- User Management Script ---"
    echo "1. Add User"
    echo "2. Delete User"
    echo "3. Modify User"
    echo "4. Assign Permissions"
    echo "5. Assign Groups"
    echo "6. Generate User Activity Report"
    echo "7. Set Password Expiration Policy"
    echo "8. Exit"

    read -p "Choose an option: " OPTION

    case $OPTION in
        1) add_user ;;
        2) delete_user ;;
        3) modify_user ;;
        4) assign_permissions ;;
        5) assign_group ;;
        6) generate_report ;;
```

```
    7) set_password_policy ;;
    8) exit 0 ;;
    *) echo "Invalid option. Please try again." ;;
  esac
done
```

Script Explanation:

1. Log File Path

```
LOGFILE="/var/log/user_management.log"
```

- **Explanation**: This variable defines the path where the log file will be stored (`/var/log/user_management.log`). The log file keeps a record of all the actions performed by the script.

3. Logging Function

```
# Function to log actions
log_action() {
    echo "$(date +'%Y-%m-%d %H:%M:%S') - $1" >> "$LOGFILE"
}
```

- **Explanation**: The `log_action` function records a timestamped action in the log file (`/var/log/user_management.log`). ○ `date +'%Y-%m-%d %H:%M:%S'` gets the current date and time.
    - ○ `$1` is the description of the action passed to the function. ○ `>>` appends the log entry to the log file.

4. Password Strength Check Function

```
# Function to check password strength
check_password_strength() {
    local PASSWORD=$1
    if [[ ${#PASSWORD} -lt 8 || ! $PASSWORD =~ [A-Z] || ! $PASSWORD =~ [a-z] || ! $PASSWORD =~ [0-9] || ! $PASSWORD =~ [^a-zA-Z0-9] ]]; then
        echo "Weak password. Must be at least 8 characters long and include upper case, lower case, digit, and special character."
        return 1
    fi
    return 0
}
```

- **Explanation**: The `check_password_strength` function checks if the password meets certain criteria:
    - ○ At least 8 characters long. ○ Contains at least one uppercase letter, one lowercase letter, one digit, and one special character.

- If the password doesn't meet these conditions, it prints a message and returns an error (1).

5. Add User Function

```bash
# Function to add a user
add_user() {
    read -p "Enter username: " USERNAME

    # Check if the user already exists
    if id "$USERNAME" &>/dev/null; then
        echo "Error: User '$USERNAME' already exists."
        return
    fi

    # Prompt for and validate password strength
    while true; do
        read -s -p "Enter password for $USERNAME: " PASSWORD
        echo
        check_password_strength "$PASSWORD"
        if [[ $? -eq 0 ]]; then break; else echo "Please try again with a stronger password."; fi
    done

    # Optional full name
    read -p "Name (Enter to skip): " FULLNAME

    # Create user with adduser or fallback to useradd
    if command -v adduser &>/dev/null; then
        echo -e "$PASSWORD\n$PASSWORD" | sudo adduser "$USERNAME" --gecos "$FULLNAME" --disabled-password > /dev/null 2>&1
    else
        sudo useradd -m -c "$FULLNAME" -s /bin/bash "$USERNAME" > /dev/null 2>&1
        echo "$USERNAME:$PASSWORD" | sudo chpasswd > /dev/null 2>&1
    fi

    # Check for success
    if id "$USERNAME" &>/dev/null; then
        # Create mail spool for the new user
        sudo touch /var/mail/"$USERNAME"
        sudo chown "$USERNAME":"$USERNAME" /var/mail/"$USERNAME"
        sudo chmod 660 /var/mail/"$USERNAME"

        log_action "Added user: $USERNAME"
        echo "User $USERNAME added successfully."
    else
        echo "Error: Failed to create user '$USERNAME'."
    fi
}
```

- **Explanation**: The `add_user` function:
  - Prompts the admin for a username.
  - Checks if the username already exists using `id`.
  - Asks for a password and verifies its strength using `check_password_strength`.
  - If the password is valid, it creates the new user with `useradd` and logs the action in the log file.

6. Delete User Function

```bash
# Function to delete a user
delete_user() {
    read -p "Enter username to delete: " USERNAME
    if id "$USERNAME" &>/dev/null; then
        sudo userdel -r "$USERNAME"
        log_action "Deleted user: $USERNAME"
        echo "User $USERNAME deleted successfully."
    else
        echo "Error: User '$USERNAME' does not exist."
    fi
}
```

- **Explanation**: The `delete_user` function:

- Prompts the admin for a username. ○ Checks if the user exists with `id`. ○ If the user exists, it deletes the user and their home directory using `userdel -r`.
- Logs the deletion and informs the admin.

7. Modify User Function

```
# Function to modify a user
modify_user() {
    read -p "Enter current username to modify: " USERNAME

    if ! id "$USERNAME" &>/dev/null; then
        echo "Error: User '$USERNAME' does not exist."
        return
    fi

    echo "What would you like to modify?"
    echo "1. Change groups"
    echo "2. Change username"

    read -p "Choose an option (1 or 2): " MODIFY_OPTION

    case $MODIFY_OPTION in
        1)
            read -p "Enter new groups: " GRPS
            sudo usermod -G "$GRPS" "$USERNAME"
            log_action "Modified groups for $USERNAME to $GRPS"
            echo "User $USERNAME groups modified successfully."
            ;;
        2)
            read -p "Enter the new username: " NEW_USERNAME

            if id -u "$NEW_USERNAME" &>/dev/null; then
                echo "Error: The username '$NEW_USERNAME' already exists."
                return
            fi

            sudo usermod -l "$NEW_USERNAME" "$USERNAME"
            log_action "Renamed user $USERNAME to $NEW_USERNAME"
            echo "User $USERNAME renamed to $NEW_USERNAME successfully."
            ;;
        *)
            echo "Invalid option. Please try again."
            ;;
    esac
}
```

- **Explanation**: The `modify_user` function:
  - Prompts the admin for a username to modify. ○ Checks if the user exists.
  - Lets the admin choose whether to change the user's groups or rename the user. ○ Executes the modification (either `usermod -G` for groups or `usermod -l` for renaming). ○ Logs the action. **8. Assign Group Function**

```
# Function to assign user to a group
assign_group() {
    read -p "Enter username to assign group: " USERNAME
    if ! id "$USERNAME" &>/dev/null; then
        echo "Error: User '$USERNAME' does not exist."
        return
    fi

    read -p "Enter the group to assign to $USERNAME: " GROUP
    if getent group "$GROUP" > /dev/null 2>&1; then
        sudo usermod -aG "$GROUP" "$USERNAME"
        log_action "Assigned user $USERNAME to group $GROUP"
        echo "User $USERNAME successfully added to group $GROUP."
    else
        echo "Error: Group '$GROUP' does not exist."
    fi
}
```

- **Explanation**: The `assign_group` function:

- Prompts the admin for a username and group. ○ Checks if the group exists using `getent group`. ○ If the group exists, it adds the user to the group with `usermod -aG`.
- Logs the assignment.

9. Assign Permissions Function

```bash
# Function to assign directory permissions
assign_permissions() {
    read -p "Enter directory path: " DIR
    read -p "Enter username: " USERNAME
    read -p "Enter permissions: " PERMISSIONS

    # Check if the user exists
    if ! id "$USERNAME" &>/dev/null; then
        echo "Error: User '$USERNAME' does not exist."
        return
    fi

    # Check if the directory exists
    if [ ! -d "$DIR" ]; then
        echo "Error: Directory '$DIR' does not exist."
        return
    fi

    # Set ownership and permissions
    if sudo chown "$USERNAME" "$DIR" && sudo chmod "$PERMISSIONS" "$DIR"; then
        log_action "Set permissions for $DIR to $PERMISSIONS for user $USERNAME."
        echo "Permissions for $DIR set successfully."
    else
        log_action "Failed to set permissions for $DIR with $PERMISSIONS for user $USERNAME."
        echo "Error: Failed to set permissions for $DIR."
    fi
}
```

- **Explanation**: The `assign_permissions` function:
  - Prompts the admin for a directory, username, and permissions.
  - Checks if the user and directory exist.
  - Sets the ownership (`chown`) and permissions (`chmod`) on the directory for the user.
    - Logs the change or an error if it fails.

10. Generate Report Function

```bash
# Function to generate user activity report
generate_report() {
    # Output file for the report
    echo "User Activity Report:" > report.txt
    echo "---------------------------------------------------------" >> report.txt
    echo "Username         Shell           Last Login Time" >> report.txt
    echo "---------------------------------------------------------" >> report.txt

    # Get a list of users with their shells and UIDs from /etc/passwd, excluding system users and nologin users
    awk -F: '{if ($3 >= 1000 && $7 != "/usr/sbin/nologin" && $7 != "/bin/false") print $1, $7}' /etc/passwd | while read -r user shell; do
        # Get the last login information using lastlog
        last_login=$(lastlog -u "$user")

        # Check if the lastlog entry shows "Never logged in"
        if echo "$last_login" | grep -q "Never logged in"; then
            last_time="**Never logged in**"
        else
            # Extract the last login time and avoid printing "Latest"
            last_time=$(echo "$last_login" | awk '{if ($4 != "Latest") print $4, $5, $6, $7, $8}')
        fi

        # Format and write the output to the report
        printf "%-15s %-15s %-40s\n" "$user" "$shell" "$last_time" >> report.txt
    done

    # Confirmation message
    echo "Report generated: report.txt"
}
```

- **Explanation**: The `generate_report` function:
  - Creates a user activity report, listing usernames, their default shells, and their last login time.

- Uses `awk` to parse the `/etc/passwd` file and checks the last login time using `lastlog`. o Saves the report to `report.txt`.

11. Set Password Expiration Policy Function

```
# Function to set password expiration policy
set_password_policy() {
    read -p "Enter username: " USERNAME

    # Check if the user exists.
    if ! id "$USERNAME" &>/dev/null; then
        echo "Error: User '$USERNAME' does not exist."
        return
    fi

    echo "Password Policy Options:"
    echo "1. Set expiration days(ex.30)"
    echo "2. Remove password expiration"

    read -p "Choose an option: " OPTION

    case $OPTION in
        1)
            read -p "Enter days until password expiration: " DAYS
            sudo chage -M "$DAYS" "$USERNAME"
            log_action "Set password expiration for '$USERNAME' to '$DAYS' days."
            echo "Password expiration policy set for '$USERNAME'."
            ;;
        2)
            sudo chage -M -1 "$USERNAME"
            log_action "Removed password expiration for '$USERNAME'."
            echo "'$USERNAME' no longer has a password expiration policy."
            ;;
        *)
            echo "Invalid option. Please try again."
            ;;
    esac
}
```

- **Explanation**: The `set_password_policy` function:
  - Allows the admin to set or remove a password expiration policy for a user using `chage`.
  - Option 1 sets the expiration to a specified number of days. o Option 2 removes the expiration.

12. Main Menu Loop

```
# Main menu loop
while true; do
    echo ""
    echo "--- User Management Script ---"
    echo "1. Add User"
    echo "2. Delete User"
    echo "3. Modify User"
    echo "4. Assign Permissions"
    echo "5. Assign Groups"
    echo "6. Generate User Activity Report"
    echo "7. Set Password Expiration Policy"
    echo "8. Exit"

    read -p "Choose an option: " OPTION

    case $OPTION in
        1) add_user ;;
        2) delete_user ;;
        3) modify_user ;;
        4) assign_permissions ;;
        5) assign_group ;;
        6) generate_report ;;
        7) set_password_policy ;;
        8) exit 0 ;;
        *) echo "Invalid option. Please try again." ;;
    esac
done
```

- **Explanation**: The main menu loop repeatedly presents options for the admin to choose from. Depending on the choice:
  - Calls the appropriate function (e.g., `add_user`, `delete_user`).

- Continues until the admin selects the option to exit (`exit 0`).

Troubleshooting:

- **Error: User Already Exists**:
  - If trying to add a user who already exists, the script will display an error message: `Error: User 'username' already exists.` To resolve this, ensure the username is unique or modify the existing user.
- **Error: User Does Not Exist**:
  - When attempting to modify or delete a non-existent user, the script will display: `Error: User 'username' does not exist.` Make sure the username is correct.
- **Weak Password Error**:
  - If the password provided does not meet the security requirements (minimum 8 characters, must include uppercase, lowercase, digits, and special characters), the script will prompt you to re-enter a stronger password.
- **Permission Denied**:
  - If the script fails to execute certain commands (e.g., `sudo` commands), make sure the script is executed with `sudo` or the necessary permissions are granted.
- **Directory/Group Not Found**:
  - When assigning permissions or groups, if the specified directory or group does not exist, the script will display: `Error: Directory 'directory' does not exist.` or `Error: Group 'group' does not exist.` Ensure the directory or group exists before assigning.
- **Password Expiration Errors**:
  - When setting password expiration, if the user does not exist or if an invalid number of days is entered, the script will return an error and prompt you to try again.
