Signup using Shell Script

[Subject – Operating System]

Name of Student	Roll No.
Athrav Jawanjal	22101C0019
Aditya Dhadve	22101C0035

Supervisor:
Mr. Mahesh Khandke

Vidyalankar Institute of Technology

(AY 2023-24)

CERTIFICATE

This is to certify that the Mini Project entitled **"Signup using Shell Script"** is a bonafide work of **Athrav Jawanjal(23101C0019), Aditya Dhadve(23101C0035)**

Mr. Mahesh Khandke
Supervisor

Contents

Abstract

This project focuses on building a basic user authentication system using shell scripting. It involves creating three essential components: a signup script that allows users to register by providing a username, email, and password, a login script that authenticates users based on the credentials stored during signup, and a password update script that allows users to change their passwords securely. The project demonstrates key shell scripting concepts, such as reading user input, file handling, and conditional logic, within a Unix-like environment, specifically targeting Linux. The signup script collects user details and stores them in a text-based data file (user_data.txt), implementing basic checks for unique usernames and matching passwords to ensure a smooth registration process. The login script authenticates users by matching their inputted credentials against those stored in user_data.txt. If a match is found, the user is granted access; otherwise, the script indicates a login failure. The password update feature enhances security by allowing users to update their credentials when needed.

Chapter 1

Introduction

1.1 Problem Statement & Objectives

1.1.1 Problem Statement

Organizations and systems often require a way to manage user authentication for secure access. Building user signup, login, and password management functionalities is a critical task, especially in environments with limited resources or specific automation requirements. Shell scripting, being lightweight and flexible, can be used to implement these functionalities. However, the inherent simplicity of shell scripting raises concerns regarding data security, scalability, and error handling. The problem is to create a basic user authentication system using shell scripting while ensuring proper validation, secure storage, and a straightforward user experience.

1.1.2 Objectives

Implement Signup Functionality

- Create a shell script that collects user information such as username, email, and password.
- Validate the inputs to ensure proper format and enforce unique usernames.

Implement Login Functionality

- Develop a shell script that prompts users for their username and password.

Implement Password Update Functionality

- Create a mechanism for users to securely update their passwords.
- Ensure proper validation and verification before allowing password changes.

Ensure Data Security and Error Handling

- Validate user inputs to prevent injection attacks and other security risks.
- Provide meaningful error messages and handle unexpected inputs gracefully.
- Ensure proper permissions and secure storage of user data.

Test and Validate the Scripts

- Test the signup, login, and password update scripts to ensure they work as expected.
- Simulate common scenarios such as successful login, failed login, successful password update, and signup with duplicate usernames.

Chapter 2

Literature Survey

2.1 Survey of Existing/Similar System

2.1.1 Django (Python)

Django provides a complete authentication system, including user signup, login, password hashing, session management, and user permission control.

2.1.2 Ruby on Rails (Ruby)

Rails has a built-in user authentication mechanism and several gems (like Devise) that offer a wide range of authentication features.

2.1.3 Express.js (Node.js)

Express can be extended with various modules like 'passport.js' for user authentication and session management.

2.1.4 WordPress

A popular Content Management System with built-in user management and roles, allowing administrators to control access to various sections.

2.1.5 Drupal

Another Content Management System with advanced user authentication and authorization capabilities.

2.1.6 JWT (JSON Web Tokens)

JWT is a popular method for securely transmitting information, often used in web applications for user authentication.

2.1.7 Firebase Authentication

A Google service that provides user authentication with features like email/-password login, social media logins, and more.

Chapter 3

Proposed System

3.1 Architecture/ Framework

3.1.1 Components

User Data File (credentials.txt)

This file serves as the data store for user information. It typically contains lines with user data in a specific format, such as username:password. This file is accessed by both the signup and login scripts to read and write user data.

Signup Script (signup.sh)

Collects user input for creating a new account (name, email, username, password). Validates user input (e.g., checking for unique usernames, valid email format, and matching passwords).

Writes the user data to user_data.txt in a consistent format. Provides feedback on success or failure (e.g., when passwords do not match or username is already taken).

Login Script (login.sh)

Prompts users for their username and password. Authenticates users by checking if the provided credentials match any entry in credentials.txt. Provides feedback on successful login or failed login. Handles common errors, such as non-existing credentials.txt or incorrect credentials.

Password Update Script (update_password.sh)

Allows authenticated users to change their passwords securely. Verifies the user's identity by checking their current credentials. Updates the stored password in credentials.txt with the new one. Provides feedback on successful or failed password updates.

Validation and Error Handling

Validates user input for proper format and consistency. Checks for errors, such as missing data file or mismatched passwords. Provides clear error messages for various failure cases, like incorrect login credentials or duplicate usernames.

3.1.2 Data Flow

Signup Process

The signup script receives user information, validates it, and then appends the data to credentials.txt. The data is stored in a consistent format for easy retrieval and parsing during the login process.

Login Process

The login script reads user input, then checks user_data.txt to find a matching username and password. If a match is found, the login is successful. Otherwise, the login fails.

Password Update Process

The update password script prompts for current username and password for verification. If authentication is successful, it prompts for the new password and confirmation. Updates the credentials.txt file by replacing the old password with the new one.

3.2 Algorithm and Process Design

3.2.1 Signup Algorithm

1. Display a welcome message for the signup process.
2. Prompt the user to enter their name and store it in a variable (name).

3. Prompt the user to enter their email address and store it in a variable (email).
4. Validate the email format (e.g., using a regular expression).
    - If the email is invalid, display an error message and return to step 3.
5. Prompt the user to enter their username and store it in a variable (username).
    - Check if the username is unique by searching in credentials.txt.
    - If the username is not unique, display an error message and return to step 4.
6. Prompt the user to enter their password and store it in a variable (password).
7. Prompt the user to confirm their password and store it in a variable (confirm).
    - If password and confirm do not match, display an error message and return to step 5.
    - If all validations are successful, append the user data (username:password) to credentials.txt.
8. Display a success message indicating successful signup.

3.2.2 Login Algorithm

1. Check if credentials.txt exists.
    - If not, display an error message and exit.
2. Prompt the user to enter their username and store it in a variable (username).
3. Prompt the user to enter their password and store it in a variable (password).
    - Search for a line in credentials.txt that matches the pattern username:*:password.
    - If a match is found, display "Login successful" and exit with success status.

- If no match is found, display "Login failed" and exit with failure status.

3.2.3 Password Update Algorithm

1. Check if credentials.txt exists.
   - If not, display an error message and exit.
2. Prompt the user to enter their username and store it in a variable (username).
3. Prompt the user to enter their current password and store it in a variable (current_password).
   - Authenticate the user using the username and current_password.
   - If authentication fails, display an error message and exit.
4. Prompt the user to enter a new password and store it in a variable (new_password).
5. Prompt the user to confirm the new password and store it in a variable (confirm_password).
   - If new_password and confirm_password do not match, display an error message and return to step 4.
6. Update the user's password in credentials.txt by replacing the old password with the new one.
7. Display a success message indicating the password has been updated.

3.2.4 Process Design

The process design for this system involves the following steps:

1. Signup Process:
   - Collect user information.
   - Validate inputs (email format, unique username, matching passwords).
   - Store validated user data in user_data.txt.
2. Login Process:

- Prompt for username and password.
- Authenticate the user by checking credentials.txt.
- Display success or failure messages based on the authentication result.

3. Password Update Process:
   - Verify current credentials.
   - Collect and validate new password.
   - Update the stored password in credentials.txt.
   - Provide feedback on the operation's success or failure.
4. Validation and Error Handling:
   - Implement input validation for email and password.
   - Ensure unique usernames.
   - Provide clear error messages for failed validations or authentication issues.
   - Handle missing files or other unexpected errors gracefully.

3.3 Details of Hardware & Software

Oracle VM Virtual Box is a software where we can more than one operating system such as Ubuntu, Fedora, Debian, Linux etc.

Kali Linux is an open-source Debian-based Linux distribution primarily used for penetration testing and cybersecurity tasks. It also supports running shell scripts through the terminal, making it a suitable environment for automating tasks and system-level scripting. This is how the Guest OS such as Kali Linux looks like:

Shell scripting doesn't strictly require Linux; it can also be used on other Unix-like operating systems and, to some extent, on Windows. However, Linux is one of the most common environments for shell scripting due to its inherent compatibility and rich ecosystem of command-line utilities.

Linux is highly customizable, with a strong emphasis on automation through shell scripts. This makes it a preferred environment for automating tasks, managing system configurations, and building workflows.

Figure 3.1: Kali Linux Desktop Screenshot

3.4 Code

3.4.1 Code for the signup page using Shell script

```
#!/bin/bash
clear

DATA_FILE="user_data.txt"
USER_CREDENTIALS_FILE="credentials.txt"

echo "Signup"
read -p "Enter your username: " new_username
read -sp "Enter a new password: " new_password
echo ""

if grep -q "$new_username:" "$USER_CREDENTIALS_FILE";
   then
    echo "Username already exists. Please try a
       different one."
else
    echo "$new_username:$new_password" >>
       "$USER_CREDENTIALS_FILE"
    echo "Sign-up Successful! You can now log in."
fi
```

3.4.2 Code for the login page using Shell script

```
#!/bin/bash
clear

DATA_FILE="user_data.txt"
USER_CREDENTIALS_FILE="credentials.txt"

echo "Login"
read -p "Enter username: " input_user
read -sp "Enter password: " input_pass
echo ""

if grep -q "^$input_user:$input_pass$"
   "$USER_CREDENTIALS_FILE"; then
    echo "Login successfully!"
    echo "User:$input_user" >> "$DATA_FILE"
else
    echo "Login failed. Incorrect Username or password."
fi
```

3.4.3 Code for the update password page using Shell script

```
#!/bin/bash
clear

USER_CREDENTIALS_FILE="credentials.txt"

echo "Update Password"
read -p "Enter username: " username
read -sp "Enter current password: " current_password
echo ""

if grep -q "^$username:$current_password$"
   "$USER_CREDENTIALS_FILE"; then
    read -sp "Enter new password: " new_password
    echo ""
    read -sp "Confirm new password: " confirm_password
    echo ""

    if [ "$new_password" = "$confirm_password" ]; then
```

```
18          # Create a temporary file and update the
               password
19          sed
               "s/^$username:$current_password$/$username:$new_password/"
               "$USER_CREDENTIALS_FILE" > temp_file
20          mv temp_file "$USER_CREDENTIALS_FILE"
21          echo "Password updated successfully!"
22      else
23          echo "New passwords do not match. Password
               update failed."
24      fi
25  else
26      echo "Authentication failed. Incorrect username or
           password."
27  fi
```

Chapter 4

Experiment and Results

4.1 Signup page

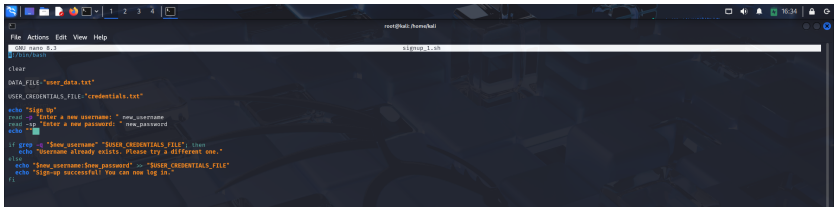

Figure 4.1: Signup Page Screenshot

4.2 Result of signup page

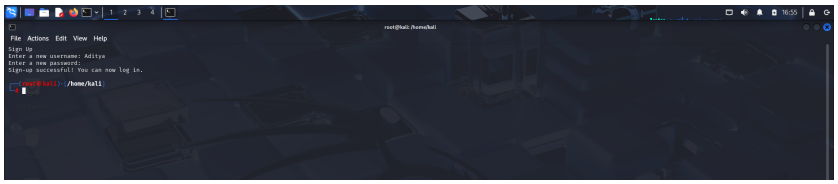

Figure 4.2: Result of Signup Page

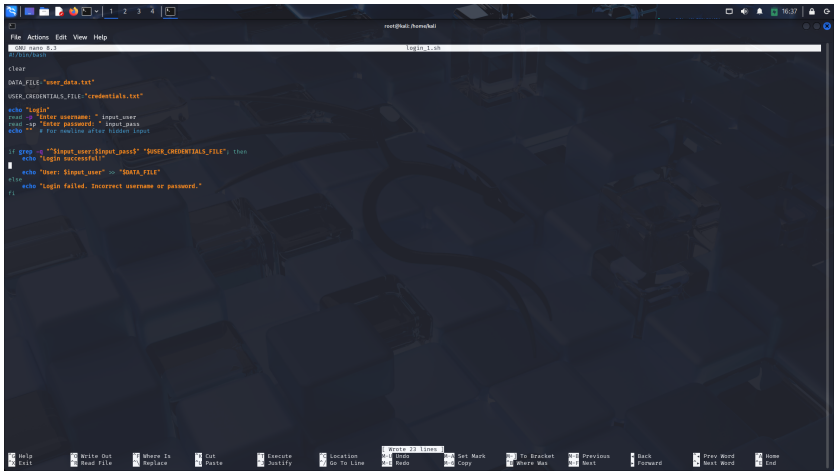

Figure 4.3: Login Page Screenshot

4.3 Login page

4.4 Result of Login page

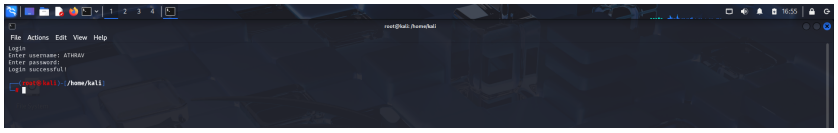

Figure 4.4: Result of Login Page

4.5 Update Password page

4.6 Result of password update page

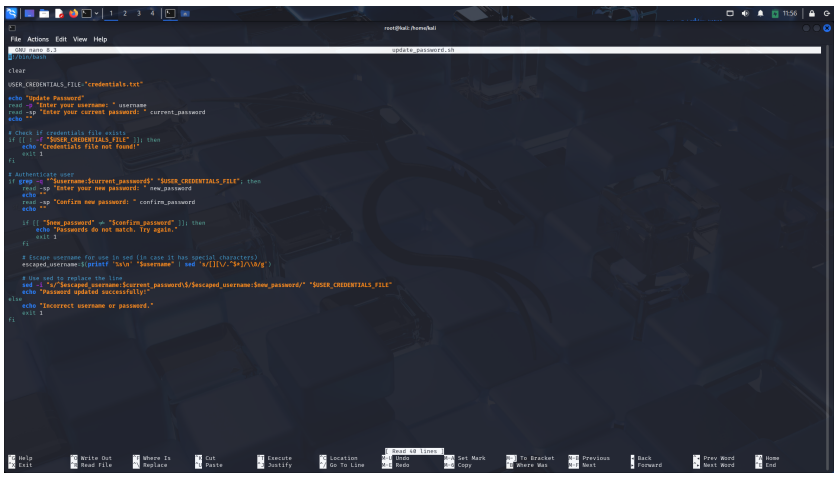

Figure 4.5: Update Password Page Screenshot

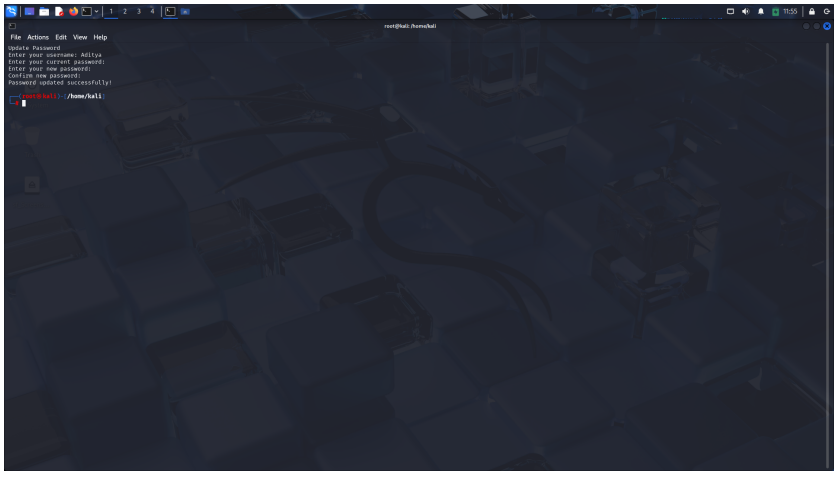

Figure 4.6: Result of Password Update Page

Chapter 5

Conclusion and Future work

This project successfully demonstrates the implementation of a basic user authentication system using shell scripting. The signup script allows users to register with unique usernames and passwords, the login script authenticates users based on stored credentials, and the password update script provides a secure way for users to change their passwords.

Key accomplishments include:

- Implementation of a signup system that validates user inputs and ensures unique usernames.
- Development of a login system that authenticates users by checking their credentials against stored data.
- Creation of a password update feature that enhances security by allowing users to change their passwords periodically.
- Storage of user information in a text file, making it easy to manage without complex database systems.

Future enhancements could include:

- Implementing password hashing to improve security.
- Adding more robust validation for user inputs.
- Developing additional features such as password recovery.
- Implementing session management for prolonged user interactions.
- Creating a more user-friendly interface using dialogue-based tools like 'whiptail' or 'dialog'.

This project demonstrates that shell scripting can be an effective tool for creating simple yet functional authentication systems, particularly in environments where simplicity and minimal resource usage are priorities.

Chapter 6

References

- Udemy Online learning platform
- Linux Shell Scripting Documentation
- Bash Manual Pages