B.M.S. COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU

UNIX SHELL PROGRAMMING
Report on

Notification and Event Reminder System (NOTIFY)

Submitted in partial fulfillment of the requirements for AAT

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

SHRAVAN SHETTY (1BM24CS423)
SACHIN KUMAR E (1BM24CS419)
PRANAY KOMMURI (1BM23CS242)
SANDESH (1BM24CS428)

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019

2024-2025

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, **SHRAVAN SHETTY (1BM24CS423), SACHIN KUMAR E (1BM24CS419), PRANAY KOMMURI (1BM23CS242), SANDESH (1BM24CS428)**, students of 3rd Semester, B.E, Department of Computer Science and Engineering, B.M.S. College of Engineering, Bangalore, hereby declare that, this AAT Project entitled "**EVENT SCHEDULER**" has been carried out in Department of CSE, B.M.S. College of Engineering, Bangalore during the academic semester September 2024 - January 2025. We also declare that to the best of our knowledge and belief, the Project report is not from part of any other report by any other students.

Signature of the Candidates

SHRAVAN SHETTY (1BM24CS423)

SACHIN KUMAR E (1BM24CS419)

PRANAY KOMMURI (1BM23CS242)

SANDESH (1BM24CS428)

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the AAT Project titled "**EVENT SCHEDULER**" has been carried out by **SHRAVAN SHETTY (1BM24CS423), SACHIN KUMAR E (1BM24CS419), PRANAY KOMMURI (1BM23CS242), SANDESH (1BM24CS428),** during the academic year 2024-2025.

Signature of the Guide

Signature of Head of the Department

Signature of Examiners with date

1. Internal Examiner

   _____

2. External Examiner

   _____

ABSTRACT

This project involves the development of a Notification and Event Reminder System (NOTIFY) using Bash scripting, designed to automate the process of scheduling event notifications and reminders. The system provides a user-friendly and efficient way to organize personal or professional events. Key functionalities of the script include

1. Secure and Accurate Input Validation:
   The script validates user inputs, including email addresses, dates, and times, ensuring accuracy and reliability. Invalid inputs prompt users for corrections, enhancing the robustness of the system

2. Event Notification Setup:
   Users can input event details such as name, description, date, time, and recipient email. A confirmation email is sent upon successful event setup, formatted with HTML for clarity and professionalism.

3. Automated Reminder Scheduling:
   The system dynamically generates temporary scripts for sending reminders. It schedules these reminders using cron jobs, starting two days before the event and running daily until the event date.

4. Email Integration:
   Notifications and reminders are sent as HTML-based emails using sendmail, providing users with visually appealing and well-organized information about their events.

This project combines the reliability of Bash scripting with practical scheduling capabilities, offering an efficient solution for event management. By leveraging features like email-based notifications, input validation, and automated reminders, NOTIFY streamlines event planning and ensures users never miss an important date.

Table of Contents

Chapter 1: Introduction

1.1 Introduction to UNIX

Overview of the UNIX Operating System

The UNIX operating system is a highly versatile and powerful system that has laid the groundwork for many modern operating systems. Developed in the early 1970s at AT&T's Bell Labs by Ken Thompson, Dennis Ritchie, and others, UNIX established the standard for operating system design with its emphasis on simplicity, modularity, and portability. Over the years, UNIX has evolved and inspired numerous derivatives, including Linux, BSD, and macOS, which are widely utilized in both academic and commercial settings today.

Key Features of UNIX

Multiuser Capability:

From the beginning, UNIX was designed to be a multiuser system, enabling multiple users to access and share system resources at the same time. This feature made it particularly suitable for large organizations where resource sharing is essential.

Multitasking:

Another key characteristic of UNIX is its multitasking capability, which allows multiple processes to run concurrently. This is accomplished through preemptive multitasking, where the operating system kernel assigns CPU time to various processes based on their priority and scheduling policies.

Portability: One of UNIX's revolutionary features is its portability. The operating system is written in the C programming language, which simplifies the process of adapting (or porting) it to different hardware platforms with minimal modifications. This aspect has played a crucial role in UNIX's widespread acceptance.

Hierarchical File System:

UNIX utilizes a hierarchical file system to organize data. Files are structured in a tree-like format, with the root directory ("/") at the top. This structure includes directories, subdirectories, and files, offering a logical and efficient method for storing and retrieving data.

Command-Line Interface (CLI):

UNIX features a robust command-line interface through its shell. The shell serves as a bridge between the user and the operating system, enabling users to execute commands, write scripts, and automate tasks. Popular shells include the Bourne shell (sh), C shell (csh), and Bash (Bash).

Security and Permissions

UNIX features a strong security model that relies on file permissions and user authentication. Each file and directory has read, write, and execute permissions that can be assigned to the owner, group, or others. This setup ensures that access to system resources is controlled and sensitive data is safeguarded.

Networking Capabilities:

UNIX was among the first operating systems to incorporate networking features into its core. This made it particularly suitable for distributed systems and client-server setups, paving the way for the internet's development.

Modularity and Simplicity:

The UNIX philosophy focuses on creating small, specialized tools that excel at performing a single task. These tools can be combined in scripts or pipelines to carry out complex operations, providing exceptional flexibility and efficiency.

Components of UNIX

Kernel:

The kernel serves as the core of UNIX. It manages hardware resources, regulates access to memory and storage, schedules processes, and oversees input/output operations. Operating at a low level, the kernel interacts directly with hardware to create a stable and secure environment for applications.

Shell:

The shell acts as a command-line interpreter, enabling users to engage with the system. It processes user commands, launches the appropriate programs, and offers scripting capabilities for automation. The shell is a vital part of UNIX, serving as a bridge between the user and the kernel.

File System:

The UNIX file system offers a cohesive method for storing and organizing data. All files and devices are represented within a single directory structure, ensuring that access is both consistent and straightforward. UNIX also supports symbolic links and hard links, allowing users to create references to files or directories.

Utilities and Commands:

UNIX comes equipped with a diverse array of built-in utilities for various tasks, including file manipulation (cp, mv, ls), text processing (grep, awk, sed), and process management (ps, kill, top). These utilities adhere to the modular design philosophy and can be combined to achieve complex results.

Applications of UNIX

UNIX is extensively utilized across various fields due to its reliability, scalability, and flexibility. Some of its main applications include:

- Server Environments: UNIX systems are frequently employed for hosting web servers, database servers, and other essential applications. Their stability and performance make them a favored option for enterprise settings.
- Software Development: UNIX offers a strong platform for software development, equipped with tools like compilers, debuggers, and version control systems that are easily accessible.
- Education and Research: UNIX is commonly used in academic institutions to teach operating system concepts, programming, and networking. Its open-source versions have made it available to students and researchers around the globe.

Legacy and Impact

UNIX has significantly shaped the design of contemporary operating systems. Its principles, such as modularity, portability, and simplicity, have been embraced by later systems, including Linux, which has become one of the most widely adopted operating systems worldwide.

Standardization efforts, like the POSIX standard, promote compatibility among UNIX-based systems, nurturing a dynamic ecosystem of tools and applications.

In conclusion, UNIX continues to be a fundamental element of the computing landscape. Its design principles, versatility, and robustness keep it relevant, even many years after its creation. Whether serving as the backbone for modern operating systems or as a resource for aspiring programmers, UNIX showcases the effectiveness of simplicity and efficiency in system design.

1.2 Motivation

Reasons for Choosing the Project.

1. Addressing Modern Organizational and Personal Challenges.

In today's fast-paced world, managing schedules and meeting deadlines has become a significant challenge. The Notification and Event Reminder System (NOTIFY) addresses this issue by offering an automated and efficient way to manage event reminders. It ensures users stay organized, reducing the likelihood of missed events and improving productivity in both personal and professional context.

2. Focus on Automation and Scheduling.

The project demonstrates the practical application of automation through dynamic integration with cron for scheduling tasks. By developing NOTIFY, users and developers alike gain an understanding of how automation tools can simplify workflows and eliminate repetitive tasks. This experience is invaluable in an era where automation plays a key role in increasing efficiency.

3. Practical Applicability.

NOTIFY has real-world applications across various domains, including personal time management, corporate event reminders, academic scheduling, and more. By building this system, the project showcases how bash scripting and UNIX utilities can be used to create tools that are directly applicable in everyday scenarios.

4. Skill Development in Bash Scripting.

The development of NOTIFY involves extensive use of Bash scripting and UNIX utilities such as cron, date, and sendmail. This fosters critical skills in system automation, file management, and task scheduling. These competencies are essential for aspiring system administrators, developers, and DevOps engineers, making the project a practical learning opportunity.

5. Emphasis on the Modularity and Design Principles.

NOTIFY is structured with modularity in mind, dividing the project into distinct components that include: • Input Validation: Ensuring the accuracy of email addresses, dates, and time inputs • Event Notification Management: Automating the scheduling and execution of reminders.

- Email Integration: Sending HTML-based confirmation and reminder emails.

6. Interactive User Interface:

The project prioritizes user experience by offering a straightforward, menu-driven interface. By providing clear prompts and intuitive navigation, NOTIFY makes technical functionalities accessible to a broader audience. This balance between usability and technical sophistication underscores the importance of user-centric software design

6. Understanding Event and Task Automation.

NOTIFY demonstrates the practical implementation of task scheduling through dynamic cron job creation. Users and developers learn to appreciate how automated systems can simplify complex workflows, ensuring timely execution of tasks without manual intervention.

7. Problem-Solving and Logical Thinking.

We project requires analyzing user requirements, designing solutions, and implementing them effectively. This process fosters critical thinking, creativity, and problem-solving skills, enabling developers to address technical challenges confidently.

8. Problem-Solving and Logical Thinking.

NOTIFY combines theoretical concepts such as UNIX principles, automation, and validation with practical implementation in Bash scripting. This holistic approach equips students and developers with a robust foundation for pursuing advanced studies or careers in system automation, software development, or project management.

9. Enhancing Time Management and Communication.

The integration of notification and reminder functionality highlights the importance of time management and timely communication. These are critical components in both personal productivity and organizational efficiency, and the project addresses them through an innovative, tech-driven solution.

Chapter 2. Methodology and Framework

Hardware and Software Requirements

Hardware Requirements

For this project, the following hardware requirements are necessary to ensure smooth execution and functionality of the encryption and decryption utility:

- Processor:
  A Minimum of a 1 GHz processor (Intel or AMD) is recommended for basic encryption operations. However, a higher processing power may be needed when handling larger files or running multiple processes simultaneously.
- RAM:
  A Minimum of 2 GB of RAM is required to ensure the smooth operation of the encryption and decryption processes. This will help to avoid system lag when multiple processes are running.
- Storage:
  At least 10 MB of available disk space is necessary for the program itself and temporary files created during encryption and decryption. Additional space will be required for encrypted files and any temporary output generated during processing.
- Operating System:
  The hardware must support an operating system compatible with Bash scripting and the necessary cryptographic libraries. UNIX-based systems (like Linux or macOS) are ideal, but the script can be adapted to other systems with minor modifications.

Software Requirements

To execute the encryption/decryption script effectively, the following software components are required:

- Operating System:
  A UNIX-based operating system, such as Linux or macOS, is required. These systems provide a native environment for Bash scripting and the required command-line utilities. While Windows users can also run the script using a Linux-like environment (e.g., through WSL – Windows Subsystem for Linux), it is best suited for UNIX-based systems.
- Bash Shell:
  The script relies heavily on Bash scripting, a default shell on most UNIX-based systems. This shell provides the necessary environment for executing the commands that handle file operations, encryption, and user interaction.
- OpenSSL:

The project uses the OpenSSL toolkit to perform the AES-256-CBC encryption and decryption. OpenSSL is a widely used open-source implementation of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols, and it includes cryptographic functions like AES encryption. OpenSSL must be installed on the system to enable the encryption and decryption functionalities.

- For Linux, OpenSSL is typically pre-installed, but if not, it can be installed using the package manager:
  sudo apt-get install openssl

- For macOS, OpenSSL is available via Homebrew:
  brew install openss

Dialog Utility:

The script uses the dialog utility to create a graphical, text-based user interface (UI) for the user. This is necessary to provide a more interactive experience for file selection, password input, and presenting operation choices. The dialog package must be installed on the system:

- On Linux, install using: sudo apt-get install dialog
- On macOS, install using Homebrew:
  brew install dialog

- Temporary File Creation:

The script uses temporary files to store passwords securely during the process. These temporary files are created using the mktemp command and are automatically removed once the operation completes. This ensures that sensitive data is not left on the system after the operation.

Chapter 3. Implementation

Source code:

```
#!/bin/bash

# Function to display a stylish header
print_header() {     zenity --info --
title="Event Reminder " \
        --text="============================\nEvent Registration &
Reminder
Scheduler\n============================="
}

# Display header
print_header

# Collect multiple inputs in a single window using zenity
form_output=$(zenity --forms --title="Event Registration & Reminder
Scheduler" \
  --text="Please enter the details of the event" \
  --add-entry="Event Name" \
  --add-entry="Email" \
  --add-calendar="Event Date" \
  --add-entry="Event Time (HH:MM AM/PM)" \
  --add-entry="Description")

# Check if the user pressed Cancel or closed the
window if [ $? -ne 0 ]; then     zenity --error --
title="Event  Scheduler" \         --text="Form
canceled. Exiting script."     exit 1 fi

# Extract the individual fields from the form output
IFS="|" read -r event_name email event_date event_time description
<<<"$form_output"

# Validate email format
if [[ ! "$email" =~ ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$ ]];
then     zenity --error --title="Event Registration & Reminder Scheduler" \
--text="Invalid email address provided. Please try again."     exit 1 fi
```

```
# Convert event time to 24-hour format
event_time_24=$(date -d "$event_time" +"%H:%M")

# Convert event date and time to seconds since epoch
event_date_secs=$(date -d "$event_date $event_time_24" +%s)

# Calculate the date for 2 days before the event
start_date=$(date -d "$event_date -2 days" +%Y-%m-%d)

# Function to send email
send_email() {
    subject="Reminder: $event_name"     body="Hello,\n\nThis is
a reminder for your event:\n\nReminder:
$event_name\nDescription: $description\nDate: $event_date\nTime:
$event_time\n\nBest Regards,\nReminder Script"

    # Send the email using msmtp
    echo -e "To: $email\nSubject: $subject\n\n$body" | msmtp --debug -
from=default -t

    # Log email delivery
status     if [ $? -eq 0 ];
then
        zenity --info --title="Reminder Scheduler" --text="Email
sent successfully to $email."     else
        zenity --error --title="Reminder Scheduler" --text="Failed to
send email. Check ~/.msmtp.log for details."     fi
}

# Confirmation email for adding the reminder subject="Reminder
Added to System" body="Hello,\n\nYour reminder has been
successfully added:\n\nReminder:
$event_name\nDescription: $description\nDate: $event_date\nTime:
$event_time\n\nYou will receive daily reminders starting two days before
the event.\n\nBest Regards,\nReminder Script"
echo -e "To: $email\nSubject: $subject\n\n$body" | msmtp --debug --
from=default
-t

# Create a valid script filename
safe_event_name=$(echo "$event_name" | tr -d '[:space:]' | tr -cd
'[:alnum:]._')
cron_script="/tmp/send_email_${safe_event_name}.sh"
```

```
# Create the cron script
cat <<EOL >
"$cron_script"
#!/bin/bash
current_date_secs=\$(date +%s)
if [ \$current_date_secs -gt $event_date_secs ]; then
    crontab -l | grep -v "$cron_script" | crontab -  # Remove cron job
after event date
    rm -f "$cron_script"                                # Clean up the
script     exit 0 fi
$(declare -f send_email)
send_email
EOL

# Make the script executable
chmod +x "$cron_script"

# Schedule the cron job to run daily from 2 days before the event
cron_time=$(date -d "$start_date" +"%M %H")  # Schedule at the same time as
START_DATE
(crontab -l 2>/dev/null; echo "$cron_time * * * bash $cron_script") |
crontab -
# Confirmation message with Zenity
zenity --info --title="Event Registration & Reminder Scheduler" \
    --text="Event and reminder scheduled successfully!\n\nDetails:\nEvent
Name:
$event_name\nEmail: $email\nEvent Date: $event_date\nEvent Time:
$event_time\nDescription: $description\n\nYou will receive reminders two
days before the event."
^C
shravan@shravanshetty:~$ chmod +x demo3_singlewindow
shravan@shravanshetty:~$ ./demo3_singlewindow
./demo3_singlewindow: line 6: --text==============================\nEvent
Registration & Reminder Scheduler\n==============================: command
not found ^C
shravan@shravanshetty:~$ cat>demo3_singlewindow
#!/bin/bash

# Function to display a stylish header
print_header() {
    zenity --info --title="Event Reminder " \
        --text="==============================\nEvent
```

```
Scheduler\n============================="
}

# Display header
print_header

# Collect multiple inputs in a single window using zenity
form_output=$(zenity --forms --title="Event Registration & Reminder
Scheduler" \
  --text="Please enter the details of the event" \
  --add-entry="Event Name" \
  --add-entry="Email" \
  --add-calendar="Event Date" \
  --add-entry="Event Time (HH:MM AM/PM)" \
  --add-entry="Description")

# Check if the user pressed Cancel or closed the
window if [ $? -ne 0 ]; then
    zenity --error --title="Event
Scheduler" \         --text="Form canceled.
Exiting script."     exit 1 fi

# Extract the individual fields from the form output
IFS="|" read -r event_name email event_date event_time description
<<<"$form_output"

# Validate email format
if [[ ! "$email" =~ ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$ ]];
then     zenity --error --title="Event Registration & Reminder Scheduler" \
        --text="Invalid email address provided. Please try
again."     exit 1 fi

# Convert event time to 24-hour format
event_time_24=$(date -d "$event_time" +"%H:%M")

# Convert event date and time to seconds since epoch
event_date_secs=$(date -d "$event_date $event_time_24" +%s)

# Calculate the date for 2 days before the event
start_date=$(date -d "$event_date -2 days" +%Y-%m-%d) #
Function to send email send_email() {
    subject="Reminder: $event_name"     body="Hello,\n\nThis is
a reminder for your event:\n\nReminder:
$event_name\nDescription: $description\nDate: $event_date\nTime:
```

```
$event_time\n\nBest Regards,\nReminder Script"

    # Send the email using msmtp
    echo -e "To: $email\nSubject: $subject\n\n$body" | msmtp --debug -
from=default -t

    # Log email delivery
status     if [ $? -eq 0 ];
then
        zenity --info --title="Reminder Scheduler" --text="Email
sent successfully to $email."     else
        zenity --error --title="Reminder Scheduler" --text="Failed to
send email. Check ~/.msmtp.log for details."     fi
}

# Confirmation email for adding the reminder subject="Reminder
Added to System" body="Hello,\n\nYour reminder has been
successfully added:\n\nReminder:
$event_name\nDescription: $description\nDate: $event_date\nTime:
$event_time\n\nYou will receive daily reminders starting two days before
the event.\n\nBest Regards,\nReminder Script"
echo -e "To: $email\nSubject: $subject\n\n$body" | msmtp --debug --
from=default
-t

# Create a valid script filename
safe_event_name=$(echo "$event_name" | tr -d '[:space:]' | tr -cd
'[:alnum:]._')
cron_script="/tmp/send_email_${safe_event_name}.sh"

# Create the cron script
cat <<EOL >
"$cron_script"
#!/bin/bash
current_date_secs=\$(date +%s)
if [ \$current_date_secs -gt $event_date_secs ]; then     crontab -l |
grep -v "$cron_script" | crontab -  # Remove cron job after event date
    rm -f "$cron_script"                            # Clean up the script
    exit
0 fi
$(declare -f send_email)
send_email
EOL
```

```
# Make the script executable
chmod +x "$cron_script"

# Schedule the cron job to run daily from 2 days before the event
cron_time=$(date -d "$start_date" +"%M %H")  # Schedule at the same time as
START_DATE
(crontab -l 2>/dev/null; echo "$cron_time * * * bash $cron_script") |
crontab -
# Confirmation message with Zenity
zenity --info --title="Event Registration & Reminder Scheduler" \
```

List of Unix Commands used with Syntax and Usage:

File and Script Handling Commands

- #!/bin/bash: Specifies that the script should be executed using the Bash shell.
- chmod +x: Makes the temporary script executable.
- mktemp: Creates a unique temporary file for the event reminder script.
- cat <<EOL >: Writes content to a file using a here-document. Text Formatting and Display Commands
- echo -e: Prints text with escape sequences (e.g., colors and newlines).
- \033[1;XXm: ANSI escape codes for text formatting (e.g., colors and bold).

Input Validation and Regular Expressions

- [[...]]: Used for conditional expressions like regular expression matching.
- if [[! "$1" =~ ...]]; then: Validates email format using a regular expression. Date and Time Handling
- date -d: Converts date/time strings into various formats or calculates offsets.
- date -d "$EVENT_DATE $EVENT_TIME_24" +%s: Converts the event date and time into a Unix timestamp.
- date -d "$START_DATE $EVENT_TIME_24" +"%M %H": Extracts the minute and hour in the required format for a cron job.

Email Handling

- sendmail: Sends emails using the sendmail command.
- echo -e "Subject:...\nContent-Type:... | sendmail ...": Sends a formatted email message with subject and body content. Cron Job Scheduling

- crontab -l: Lists the current user's cron jobs.
- (crontab -l 2>/dev/null; echo "...") | crontab -: Appends a new cron job to the current user's crontab.

Exit Status Checking

- if [[$? -ne 0]]; then: Checks the exit status of the previous command to determine success or failure.
- &>/dev/null: Redirects both stdout and stderr to null, suppressing output.

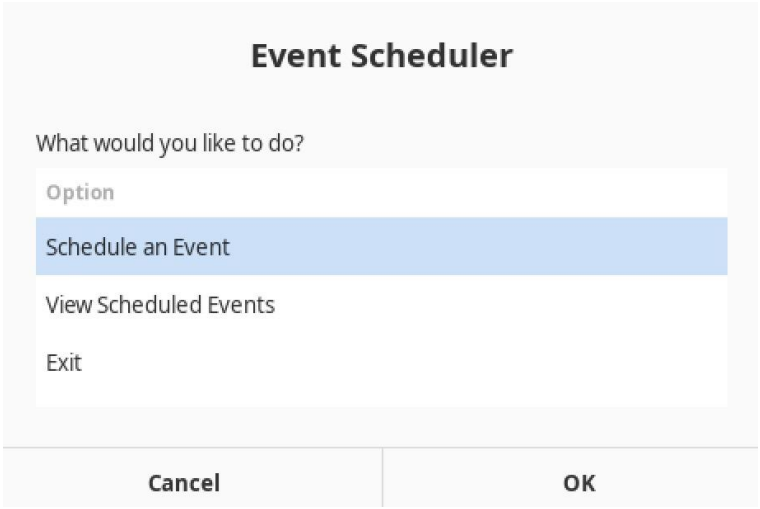
User Input and Read

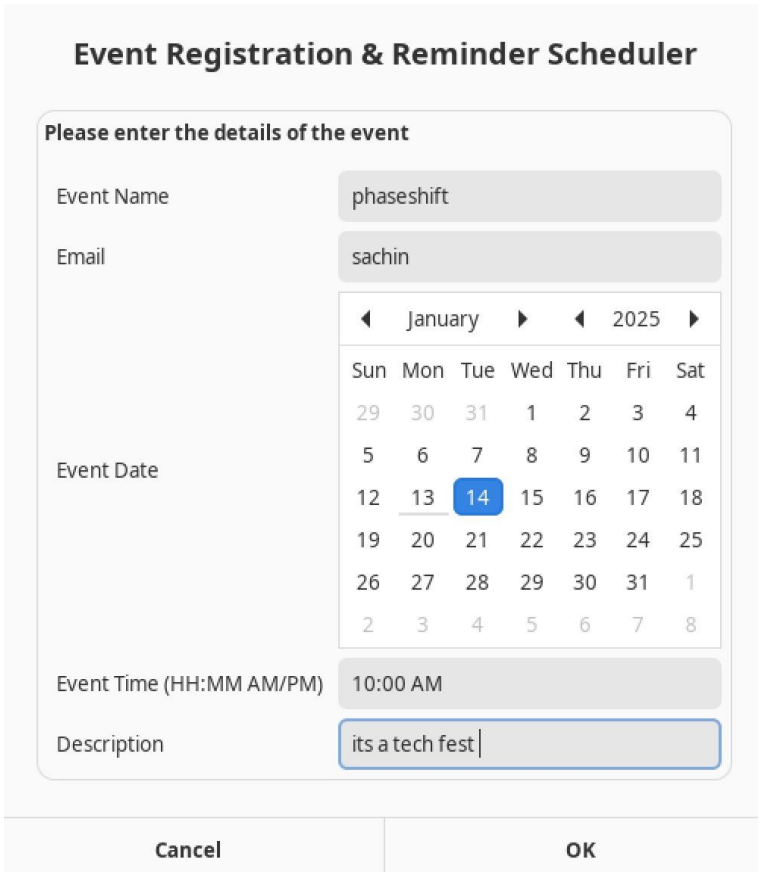
- read -p: Reads input from the user with a prompt.
- while true; do ... break; done: Loops until valid input is provided. Function Definitions
- function_name() { ... }: Defines reusable functions like validate_email, validate_date, print_header, and send_email.
- $(declare -f function_name): Exports function definitions for use in a temporary script. HTML Content in Strings
- Content-Type: text/html; charset=UTF-8: Specifies that the email body contains HTML.
- <html> and <body> tags: Formats the email with HTML content for better readability.

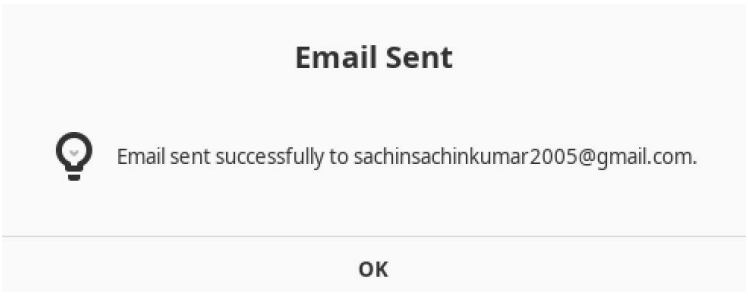
Miscellaneous

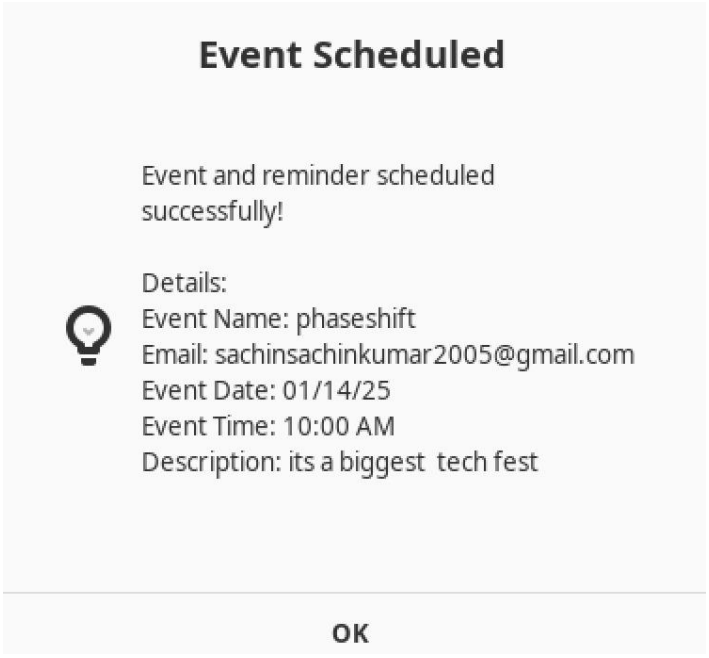
- return: Exits a function with a specific status.
- break: Exits a loop.

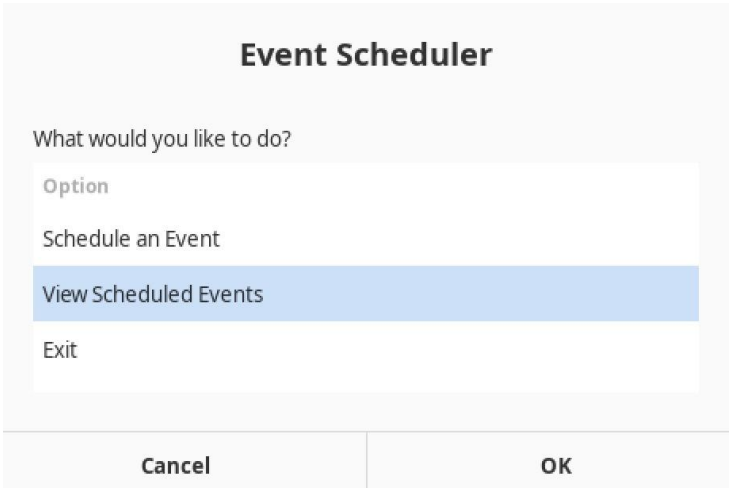
Chapter 4. Result and Analysis

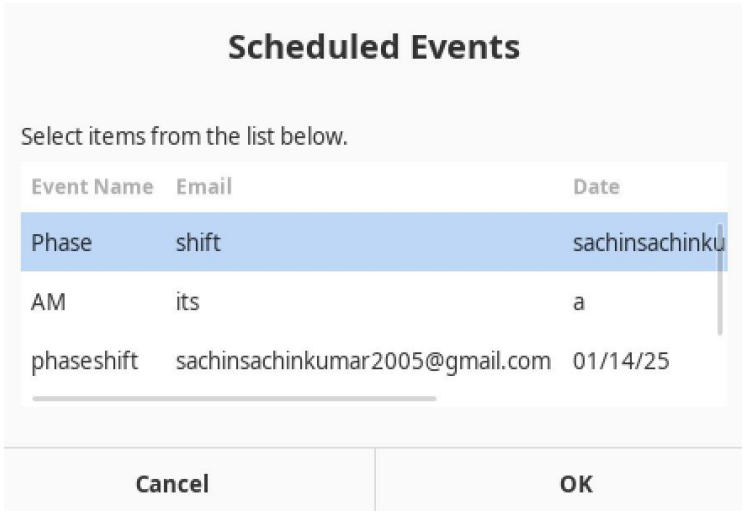







Event Scheduled
Event and reminder scheduled successfully!
Details:
Event Name: phaseshift
Email: sachinsachinkumar2005@gmail.com
Event Date: 01/14/25
Event Time: 10:00 AM
Description: its a biggest tech fest
OK


Event Scheduler
What would you like to do?
Option
Schedule an Event
View Scheduled Events
Exit
Cancel
OK


Scheduled Events
Select items from the list below.
Event Name Email Date
Phase shift sachinsachinku
AM its a
phaseshift sachinsachinkumar2005@gmail.com 01/14/25
Cancel
OK

Chapter 5. Conclusion and Future Enhancement

5.1 Conclusion

In conclusion, this project successfully implements an email notifier system using Bash scripting. The project demonstrates how automation can be leveraged to simplify event notification processes, ensuring timely updates and reminders for users.

Key features include:

- Input Validation: Ensuring accuracy through validation of email, date, and time formats.
- Dynamic Scheduling: Automated scheduling of reminders using cron jobs.
- Email Integration: Seamless integration of email notifications with formatted content for enhanced user experience.

5.2 Future Enhancement

To further enhance the utility and versatility of the email notifier system, several improvements can be implemented. One significant enhancement is adding support for multiple notification channels, such as SMS, push notifications, or messaging platforms like WhatsApp and Telegram, providing users with more flexibility. Integration with popular calendar systems like Google Calendar or Microsoft Outlook could allow automatic event imports and seamless synchronization of reminders. Additionally, enabling batch event scheduling would streamline the management of multiple notifications, making the system more efficient. Developing a graphical user interface (GUI) would improve accessibility for non-technical users, offering an intuitive way to schedule and manage notifications. Advanced scheduling features, such as recurring events and customizable reminder intervals, would further increase the system's flexibility. Enhancing security by encrypting stored data and credentials would ensure sensitive information remains protected. Lastly, implementing robust error-handling and logging mechanisms would improve the system's reliability and provide better monitoring capabilities. These enhancements would transform the email notifier into a more powerful, user-friendly, and secure solution, catering to a broader audience and addressing more complex notification requirements.

References

1. OpenSSL Documentation
   Official OpenSSL documentation providing detailed information on cryptographic functions, algorithms, and usage:
   https://www.openssl.org/docs/

2. Dialog Utility Documentation
   Comprehensive guide on the dialog utility for creating graphical command-line interfaces: https://invisible-island.net/dialog/
3. Linux Manual Pages (man pages)
   Detailed manual pages offering explanations of system commands, tools, and functions used in UNIX/Linux systems: https://man7.org/linux/man-pages/
4. Arch Wiki - Bash Tips and Tricks
   A practical resource for advanced Bash scripting techniques, including best practices and common use cases: https://wiki.archlinux.org/title/Bash#Tips_and_tricks
5. GNU Bash Reference Manual
   Official GNU manual for Bash, covering syntax, built-in commands, and scripting techniques: https://www.gnu.org/software/bash/manual/bash.html
6. Cron and Scheduling Documentation
   Guides on configuring and managing cron jobs for task scheduling in UNIX/Linux systems: https://man7.org/linux/man-pages/man5/crontab.5.html
7. Email Utility Documentation
   Manuals and guides for email tools like mail and sendmail for sending automated email notifications: https://man7.org/linux/man-pages/man1/mail.1.html
8. Bash Script Examples
   Examples and best practices for writing secure and efficient Bash scripts: https://www.gnu.org/software/bash/manual/bash.html