Jaypee Institute of Information Technology, Sector - 62, Noida

B.Tech CSE III Semester

Unix Lab PBL Report

TUI Notes Manager

Submitted to

Dr. Amarjeet Kaur
Dr. Amit Mishra
Dr. Meenal

Submitted by

Harsh Sharma	2401030232	B5
Karvy Singh	2401030234	B5
Rudra Kumar Singh	2401030237	B5

Letter of Transmittal

Dr. Amarjeet Kaur
Dr. Amit Mishra
Dr. Meenal

Department of Computer Science & IT

Subject: Submission of Project "TUI Notes Manager"

Respected Madam/Sir,

We are pleased to submit our project titled *"TUI Notes Manager"* as part of the Unix Laboratory course. This report documents the motivation, design, implementation details, and learning outcomes of the project.

We have endeavoured to cover all the major Unix concepts taught in the lab, including shell scripting, text processing utilities, pipes and redirection, signal handling with traps, terminal control, colour handling using `tput`, and job scheduling with both `crontab` and the `at` command. The project integrates these concepts into a practical Text User Interface (TUI) application for managing notes and reminders.

Thank you for your guidance and the opportunity to work on this project.

Sincerely,

Harsh Sharma (2401030232)
Karvy Singh (2401030234)
Rudra Kumar Singh (2401030237)

Date: November 20, 2025

Contents

1 Introduction

In day-to-day academic work, students frequently need to maintain small text notes such as to-do lists, code snippets, ideas, and reminders. Most existing solutions are graphical or web based and may be heavy-weight for quick usage from a Unix terminal environment.

The **TUI Notes Manager** is a shell-script-based terminal application that allows users to create, browse, search, and edit notes, as well as schedule reminder notifications using either `crontab` (for repeating tasks) or the `at` command (for one-off tasks). The entire application is implemented as a single Bash script (`notes.sh`) and operates inside a full-screen TUI, making it convenient for users who primarily work in the terminal.

From an academic perspective, this project is designed to showcase key Unix and shell programming concepts introduced in the Unix Lab:

- Shell scripting and modular functions.
- Pipes and input/output redirection.
- Usage of standard Unix text processing tools such as `grep`, `sed`, and `cal`.
- Terminal handling using `tput`, including alternate screen buffers, cursor control, and colours.
- Signal handling using `trap` (for `EXIT` and `WINCH`).
- Persistent background scheduling using `crontab` for repeating reminders.
- Non-repetitive reminders scheduled using the `at` command.

2 Problem Statement and Objectives

The goal of the project is to design and implement a lightweight notes manager that runs entirely in a Unix terminal and meets the following objectives:

1. Provide a user-friendly TUI for listing, viewing, and editing notes stored as plain text files in a dedicated directory (`$HOME/notes`).
2. Allow quick creation, renaming, and deletion of notes using simple keyboard shortcuts.
3. Support fast searching across all notes using `grep`, with interactive navigation of search results.
4. Integrate a calendar view to navigate dates and attach tasks/reminders to specific days.
5. Use `crontab` to schedule repeating reminder notifications at specified dates and times.
6. Use the `at` command to schedule non-repeating reminders.
7. Demonstrate robust Unix shell scripting practices such as `set -euo pipefail`, traps for cleanup and window resizing, safe handling of user input, and graceful fallback when colours are not supported.

3 Software Requirements

- Unix-like operating system (GNU/Linux recommended).
- Bash shell (the script uses Bash-specific features).
- A terminal emulator supporting ANSI escape codes and colours.
- A text editor such as `nvim`, `vim`, or `vi`.
- `tput` (from `ncurses`) for terminal capability queries and colour handling.
- `crontab` for scheduling repeating tasks.
- `at` for scheduling single-run, non-repeating tasks.
- `notify-send` (freedesktop notifications) for desktop alerts.
- Standard Unix utilities: `grep`, `sed`, `tput`, `cal`, `id`, and `date`.

4 System Design

4.1 Overall Architecture

The TUI Notes Manager is implemented as a single Bash script, `notes.sh`. The script is logically divided into the following components:

- **Initialization**: Setting shell options, preparing directories, selecting the text editor, and configuring terminal settings.
- **Colour Setup**: Detecting terminal colour capabilities with `tput colors` and initialising colour variables for titles, status lines, previews, and selections, with automatic fallback to monochrome if colours are unavailable.
- **Layout and Rendering**: Computing terminal dimensions and drawing panels using `tput`.
- **Modes of Operation**: Three main modes—*browse*, *search*, and *calendar*—each managed by its own event loop.
- **Note Management**: Functions to list, create, rename, delete, and preview note files.
- **Search and Filtering**: Running `grep` recursively to search within notes and displaying matching files.
- **Reminder Management**: Functions to list, add, and delete tasks that are stored either as `crontab` entries (for repeating tasks) or as `at` jobs (for one-off tasks).
- **Signal Handling and Cleanup**: Traps for restoring the terminal state and handling window resize events.

4.2 Data and File Layout

- All notes are stored as plain text files in a directory:

  `$HOME/notes`

- Filenames are generated from user-provided titles by the `slugify` function, which:
  - Converts letters to lowercase.
  - Replaces non-alphanumeric characters with hyphens.
  - Ensures there is always a non-empty slug (default `note`).
- A separate directory is used to cache metadata:

  `$HOME/.local/state/notes`

  which stores the currently edited note path.
- Tasks and reminders share a common textual marker of the form:

  `# TASK:YYYY-MM-DD:Task description`

  - For repeating tasks, this marker appears in the user's `crontab` lines.
  - For one-off tasks, the marker appears inside the job body scheduled with `at`.

  This unified marker allows the script to present both kinds of tasks together in the calendar view.

4.3 User Interaction Flow

The high-level flow of the program is shown in Figure 1. It starts by initialising the environment, then enters the alternate screen and main loop. Depending on the current mode, key presses invoke different actions such as browsing notes, searching, or managing tasks (including choosing between repeating and one-off reminders). On exit, the terminal is restored to its original state.

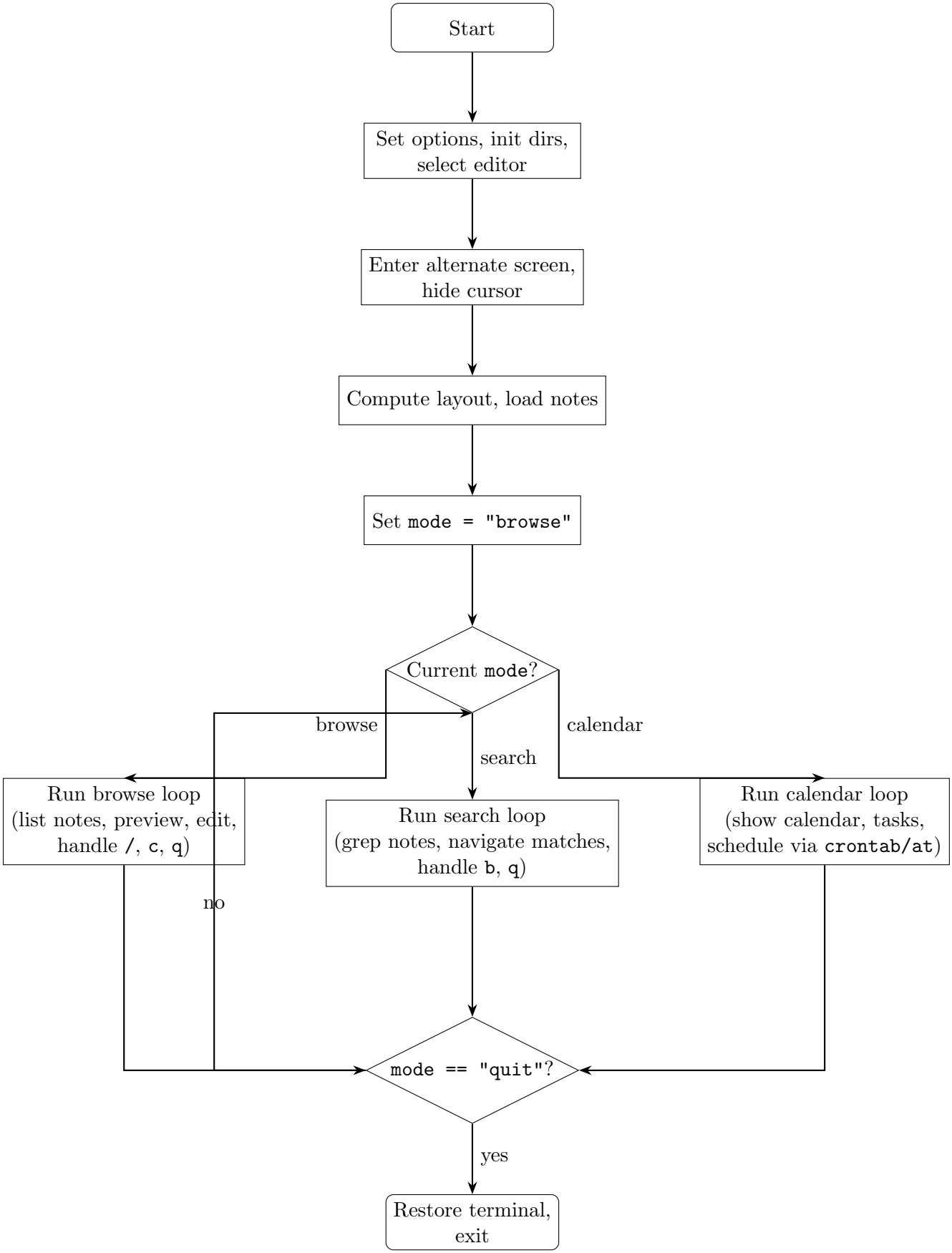

Figure 1: High-level control flow of the TUI Notes Manager

5 Implementation Details

5.1 Shell Options and Initialization

The script begins with:

- `#!/bin/bash` to ensure Bash is used.
- `set -euo pipefail` to enable strict error handling:
  - `-e` stops execution on errors.
  - `-u` treats unset variables as errors.
  - `pipefail` ensures pipeline failures are detected.

The notes directory is prepared using:

```
notesDir="$HOME/notes"
mkdir -p "$notesDir"
```

The script selects an editor based on the `EDITOR` environment variable and falls back to `nvim`, `vim`, or `vi` using `command -v`.

A cache directory at `$HOME/.local/state/notes` stores the path of the currently open note.

5.2 Terminal Handling, Colours, and Layout

The script uses `tput` to control cursor visibility, clear the screen, query colour capabilities, and compute terminal dimensions:

- `tput cols`, `tput lines` to get width and height.
- `tput colors` to detect whether the terminal supports colours and how many.
- Alternate screen buffer is enabled with `ESC[?1049h` and disabled with `ESC[?1049l`.
- The cursor is hidden during TUI operation and restored on exit.

When colours are available (at least 8), the script defines colour variables using `tput setaf` and `tput setab` for:

- Panel titles (cyan, bold).
- Status bar (green foreground).
- Preview text (distinct colour).
- Selection highlighting (blue background with white foreground).

If colours are not supported, the script falls back to monochrome attributes such as `tput bold` and `tput rev`.

The screen is divided into two panels:

- Left panel: list of notes or search results.
- Right panel: preview of the selected note or tasks for the selected date.

The `layoutDim` function computes the dimensions of these panels dynamically based on terminal size.

5.3 Mode Management

The global variable `mode` controls which event loop is active. Possible values are:

- `"browse"`: default mode for listing and editing notes.
- `"search"`: search results navigation mode.
- `"calendar"`: calendar and task management mode.
- `"quit"`: signals that the main loop should terminate.

Each mode has its own loop function:

- `loop_browse`
- `loop_search`
- `loop_calendar`

The main loop repeatedly calls the appropriate function until `mode = "quit"`.

5.4 Key Input and Navigation

The script reads raw key input using a custom `read_key` function that handles escape sequences for arrow keys:

- Up/Down arrows or `k`/`j` move the selection.
- Page up/down sequences scroll faster in the notes list.
- `e` or Enter edits the current note.
- `n` creates a new note or task (depending on mode).
- `r` renames a note.
- `d` deletes a note or task.
- `/` initiates search.
- `c` switches to calendar mode.
- `b` returns from search or calendar to browse mode.
- `q` quits the application.

The helper function `moveSelection` ensures that navigation wraps around and keeps the selected item within the visible window.

5.5 Note Management

The `list_files` function lists all note files in reverse chronological order using `ls -t`. The filenames are stored in the `presentFiles` array using `mapfile`.

Key operations:

- **New note** (`newNote`):
  - Prompts for a title.
  - Generates a slug using `slugify`.
  - Opens the corresponding file in the editor.
- **Rename note** (`renameNote`): prompts for a new title and renames the file.
- **Delete note** (`deleteNote`): confirms with the user and removes the file.
- **Edit note** (`openEditor`): temporarily leaves the alternate screen, launches the editor, and then restores the TUI.

For previewing, the script uses:

```
sed -n '1,200p'
```

to show the first few lines of the selected note, rendered within the right-hand panel, using a dedicated preview colour when available.

5.6 Search Implementation

When the user presses `/`, the script reads a search query and calls `runSearch`:

- `grep -RIn` is used to search recursively within the notes directory.
- Each matching line is parsed to extract the filename and snippet.
- Results are stored in `searchedFiles` and `searchedSnips` arrays.

The search mode displays the list of matching files on the left and a preview of the selected file on the right. Pressing `e` or Enter opens the currently selected match for editing.

5.7 Calendar and Reminder Management

The calendar view uses the `cal` command to render a monthly calendar. The current day in `calendarDate` is highlighted using reverse video attributes.

The `list_tasks` function aggregates both repeating and one-off tasks:

- Cron-based tasks are obtained from `crontab -l` and filtered by the date-specific marker `# TASK:YYYY-MM-DD:`.
- One-off tasks are discovered using `atq`. For each job ID, the script inspects the job body with `at -c` and looks for the same marker.

- Each task is tagged with a prefix indicating its origin (`cron` or `at`) so that deletion can be handled correctly.

When creating a new task, `newTask`:

1. Prompts the user for the task description.
2. Prompts for the time (HH:MM).
3. Asks whether the task is repetitive.
4. If the user chooses a repeating task, the script appends a suitable line to the user's `crontab`, including the `notify-send` command and the `# TASK:...` marker.
5. If the user chooses a one-off task, the script constructs a combined date/time string, validates it, and pipes a `notify-send` command with the marker into `at -t`.

The `deleteTask` function handles both kinds of tasks:

- For cron tasks, the relevant line is removed from `crontab`.
- For at-based tasks, the corresponding job is removed using `at -d` or `atrm`.

This design demonstrates real-world usage of `crontab`, `at`, environment variables like `XDG_RUNTIME_DIR` and `DBUS_SESSION_BUS_ADDRESS`, and interaction with the desktop notification system.

5.8 Signal Handling and Robustness

The script uses `trap` to handle important signals:

- `trap 'on_exit' EXIT`: ensures that the alternate screen is disabled and the cursor is restored even if the script exits unexpectedly.
- `trap 'resize' WINCH`: handles terminal window resize events by recomputing layout dimensions and forcing a redraw.

These traps improve user experience and maintain terminal integrity.
In addition, the script safely handles user input:

- Temporarily enabling echo with `stty echo` while reading titles, queries, or task descriptions.
- Restoring previous terminal settings after input.

6 Unix Concepts Demonstrated

6.1 Shell Scripting and Modular Design

The project makes extensive use of functions, global state variables, and arrays. It demonstrates:

- Function definitions and reuse.
- Global variables for managing state.
- Arrays with `mapfile` and indexed access.

6.2 Pipes and I/O Redirection

The script uses pipes and redirection in several places:

- `list_files` uses `ls` piped into a `while read` loop.
- `runSearch` uses `grep -RIn` piped into a parser.
- `crontab -l 2>/dev/null | grep ...` to filter tasks.
- Inspection of `at` jobs is done by piping the output of `at -c` into `grep`.

6.3 Text Processing Utilities

The project uses Unix text processing commands:

- `grep` for search and filtering.
- `sed` for preview and text substitution (for highlighting the current day in the calendar).
- `tr` and `sed` in `slugify` for generating filenames.

6.4 Terminal Control and Colour Handling

Using `tput`, the script:

- Moves the cursor to arbitrary positions with `tput cup`.
- Clears specific lines or the whole screen.
- Configures text attributes such as bold and reverse video.
- Queries terminal colour support and sets foreground and background colours for different UI elements.

The application also falls back gracefully to monochrome where colour is not available, improving portability.

6.5 Signal Handling with `trap`

Signals are used to:

- Cleanly restore the terminal when the script exits.
- Respond to window size changes without restarting the program.

6.6 Job Scheduling with `crontab` and `at`

The calendar mode shows a practical application of both `crontab` and `at`:

- Programmatically adding cron entries for repeating tasks.
- Scheduling one-off notifications with `at -t`.
- Parsing and deleting specific entries or jobs.
- Using comments as structured metadata to associate each job with a date and description.

7 Limitations and Future Work

Some known limitations of the current implementation are:

- The interface is purely keyboard based and may not be intuitive for first-time users without documentation.
- Reminders rely on `notify-send` and a graphical session; on a pure text console, notifications may not appear.
- Tasks are stored only in `crontab` comments and `at` job bodies and are not indexed or grouped by project.
- Notes are plain text and do not currently support tagging or rich formatting.

Potential future enhancements include:

- Adding support for note tags and filtering by tag.
- Implementing basic markdown preview in the TUI.
- Supporting multiple note collections (for example, academic, personal, projects).
- Exporting and importing tasks and notes as archive files.
- Adding configurable colour themes for better visual clarity.

8 Conclusion

The TUI Notes Manager project successfully demonstrates how Unix and shell scripting concepts can be combined to create a practical and user-friendly terminal application. By integrating note management, full-screen TUI rendering, search functionality, a calendar, and reminder scheduling via both `crontab` and `at`, the project reflects the core topics of the Unix Lab.

Beyond fulfilling academic objectives, the tool is genuinely useful for everyday workflow, especially for users who prefer staying within the terminal. The project strengthened our understanding of Bash scripting, Unix utilities, terminal handling, colour control, and background job scheduling, and provided experience in designing and debugging interactive command-line applications.

References

- *Bash Reference Manual*, GNU Project.
- Linux manual pages: `bash(1)`, `grep(1)`, `sed(1)`, `tput(1)`, `crontab(1)`, `at(1)`, `notify-send(1)`, `cal(1)`.
- *Advanced Bash-Scripting Guide*, The Linux Documentation Project.