ChatGPT

1-Month Roadmap: Intelligent Unix Shell

The team will build a C/POSIX-based shell core that loops to read, parse, and execute commands [1], augmented by a Python-based ML suggestion engine. The shell and Python components will communicate via Unix domain sockets for low-overhead IPC [2]. Command histories will be logged in SQLite (using `sqlite3_open`, `sqlite3_exec`, etc.) for training and suggestions [3]. Roles are distributed so that Anshika leads core systems programming, Akriti leads machine learning (and documentation), Ayush leads SQLite integration and testing, and Rakhi leads IPC (with each member assisting across tasks). The following weekly milestones break down goals, tasks, and deliverables in detail.

Week 1: Setup and Core Design

- **Goals:** Project kickoff with design of system architecture. Establish development environment, repository, and coding standards. Outline shell functionality and suggestion-engine interface.
- **Tasks:**
- Set up version control, define coding style, and create initial project plan (Lead: Akriti, Assist: Rakhi).
- **Architecture design:** Define how the shell core and Python ML engine interact via a Unix domain socket [2]. Prepare interface protocol (e.g. simple text/JSON). (Lead: Rakhi, Assist: Akriti)
- **Core shell loop:** Implement the main command loop (read a line from stdin, parse it into arguments, execute the command) [1]. (Lead: Anshika, Assist: Ayush, Rakhi)
- **Command parsing:** Write a simple lexer/parser for splitting input into program and arguments (using `strtok` or similar). (Lead: Anshika, Assist: Rakhi)
- **Basic execution:** Invoke external programs with `fork()` and `execvp()`. (Lead: Anshika, Assist: Ayush)
- **SQLite setup:** Integrate SQLite in C. Write code to open/create a database and table (using `sqlite3_open` / `sqlite3_exec`) [3]. (Lead: Ayush, Assist: Anshika)
- **Initial IPC test:** Write a simple Unix-domain-socket server in Python and a C client to send a test message (verify basic IPC). (Lead: Rakhi, Assist: Akriti)
- **Documentation:** Draft outline of user guide and technical docs. (Lead: Akriti, Assist: Rakhi)
- **Deliverables:**
- Initialized code repository with basic README and design docs.
- A stub shell program with a working read/parse/execute loop [1].
- SQLite database file created (e.g. `commands.db`) with an empty history table.
- Working proof-of-concept IPC: C program sends a message, Python server echoes or logs it.
- Project plan and documentation outline (to be expanded).

Week 2: Feature Implementation and Integration

- **Goals:** Extend shell functionality (pipes, redirection, built-ins) and integrate components. Begin basic ML suggestion logic using gathered history.
- **Tasks:**
- **Built-in commands:** Implement `cd`, `exit`, and a `history` command (reading from SQLite). (Lead: Anshika, Assist: Ayush, Rakhi)

- **Pipes/redirection:** Add support for shell pipelines (using `pipe()`) and input/output redirection (`<`, `>`). (Lead: Anshika, Assist: Rakhi)
- **Background jobs:** Allow `&` to run processes in the background. (Lead: Anshika, Assist: Rakhi)
- **SQLite logging:** After each command execution, insert the command string into the SQLite history table. (Lead: Ayush, Assist: Anshika)
- **ML prototype:** In Python, read recent command history from SQLite and implement a simple suggestion algorithm (e.g. most-frequent next command or n-gram predictor). (Lead: Akriti, Assist: Ayush)
- **IPC integration:** Modify the shell to send the current (partial) command line to the Python server over UDS and display any suggestion received. Test the end-to-end flow. (Lead: Rakhi, Assist: Akriti)
- **Testing:** Write and run basic test cases for shell features (e.g. running commands, pipes, built-ins). (Lead: Ayush, Assist: all)
- **Documentation:** Update the user guide with instructions for new features (execution, built-ins, data logging). (Lead: Akriti, Assist: Rakhi)
- **Deliverables:**
- Enhanced shell supporting pipelines, I/O redirection, background jobs, and built-ins.
- Integrated SQLite history logging working; verify `history` command reads from DB.
- Python suggestion server running; shell can send a prefix and get back a suggestion string.
- Test report with results of functional tests (shell correctness, IPC messages).
- Draft user guide sections covering usage of the above features.

Week 3: Refinement and Advanced Features

- **Goals:** Improve the suggestion engine, add advanced shell features, and build robustness. Increase test coverage and refine documentation.
- **Tasks:**
- **Enhanced ML engine:** Improve the Python model (e.g. n-gram language model or simple collaborative filtering on commands). Use SQLite to query likely suggestions by prefix. (Lead: Akriti, Assist: Ayush)
- **Suggestion integration:** In the shell's prompt, show suggestions (e.g. grayed autocomplete or hint) from the ML engine in real-time. Accept suggestion on keypress (if feasible). (Lead: Rakhi, Assist: Anshika)
- **Advanced shell behavior:** Support features like wildcard expansion, environment variables, and signal handling (e.g. `Ctrl+C`). (Lead: Anshika, Assist: Rakhi)
- **Testing & validation:** Expand tests to edge cases (invalid commands, long inputs). Profile performance of SQLite queries and socket latency. (Lead: Ayush, Assist: all)
- **Documentation:** Write the remaining sections of the user guide, including installation, examples, and FAQ. Ensure clarity and completeness [4]. (Lead: Akriti, Assist: Rakhi)
- **Deliverables:**
- Finalized suggestion engine integrated into the shell, with improved accuracy.
- Full-featured shell with advanced functionality and robust error handling.
- Complete suite of test cases and a validation report (e.g. shell behaves as expected under various scenarios).
- Comprehensive user manual (clear, comprehensive, easy-to-understand [5]) and technical documentation.

Week 4: Finalization and Delivery

- **Goals:** Polish, fix bugs, optimize performance, and prepare final artifacts. Ensure everything is well-tested and documented.

- **Tasks:**
- **Bug fixes:** Address any defects found in testing (shell crashes, suggestion errors). (Lead: Anshika, Assist: Rakhi)
- **Performance tuning:** Optimize SQLite queries (use indices if needed) and ensure socket communication is non-blocking or threaded. (Lead: Ayush, Assist: Rakhi)
- **User documentation:** Final review and proofreading of user guide. Add final screenshots/ examples. (Lead: Rakhi, Assist: Akriti)
- **Final testing:** Conduct end-to-end tests (all features together). Prepare test results summary. (Lead: Ayush, Assist: all)
- **Demo preparation:** Create a demo script or presentation showcasing the shell's features. (Lead: Akriti, Assist: team)
- **Deployment pack:** Package code, documentation, and instructions for building/running. (Lead: Akriti, Assist: team)
- **Deliverables:**
- Release-ready shell executable and source code.
- Final user guide and technical documentation.
- Test summary report and any scripts used.
- Demo materials (slides or video) if required.

Role Assignment Summary

Team Member	Systems Programming (C/POSIX)	ML Component (Python)	SQLite Integration	IPC (Unix Sockets)	Testing & Validation	Documentation/ User Guide
Anshika	Lead	Assist	Assist	Assist	Assist	–
Akriti	Assist	Lead	–	–	–	Lead
Ayush	Assist	Assist	Lead	Assist	Lead	–
Rakhi	Assist	–	–	Lead	Assist	Assist

- *Lead* indicates the primary person responsible; *Assist* indicates supporting roles. Roles may shift slightly as the project evolves, but this table outlines the planned distribution.

Sources: The shell's design follows a classic read-parse-execute loop [1]. Unix domain sockets will be used for efficient same-machine IPC [2]. SQLite usage is based on the standard C API (e.g. `sqlite3_open`) [3]. Good documentation should be clear and comprehensive [5].

---

[1] Tutorial - Write a Shell in C • Stephen Brennan
https://brennan.io/2015/01/16/write-a-shell-in-c/

[2] Tinkering with Unix domain sockets | Redowan's Reflections
http://rednafi.com/misc/tinkering_with_unix_domain_socket/

[3] SQLite - C/C++
https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm

[4] [5] User Documentation: The Ultimate Guide for Product Managers
https://userpilot.com/blog/user-documentation/