Command Line Interface Guidelines: A modern philosophy for the terminal

We often treat command-line tools as pure utility: scripts to be piped, args to be parsed, and exit codes to be checked. But as computing pioneer Alan Kay noted, the command line remains the most *versatile* corner of the computer, offering a depth GUIs cannot afford. The Command Line Interface Guidelines (CLIG) proposes a necessary modernization: shifting from a **machine-first** tradition to a **human-first** design philosophy.

The Baggage of the Past

Traditional UNIX philosophy optimized for composability and brevity—machines talking to machines. This often resulted in opaque error messages and the silent treatment on success (the classic "no news is good news" approach of `cp`). CLIG argues that today's terminal is primarily a human workspace. If a command changes the state of the system, it should explicitly tell the user what happened, much like `git push` details exactly which refs were updated, rather than just returning exit code 0.

Conversation as the Interface

The guide introduces a powerful metaphor: *conversation*. Using a CLI is a dialogue where the user proposes an intent, and the program responds. This reframes error handling not as a crash, but as a corrective turn in the conversation.

For instance, if I run `brew update jq`, the tool shouldn't just fail; it should gently suggest, *"Did you mean 'brew upgrade jq'?"* This is empathy in software design. It acknowledges that the user is learning through repeated trial-and-error.

Concrete Heuristics for Modern Tools

The document is rich with specific implementation details that distinguish a "toy" script from a professional tool:

- **Context-Aware Output (The TTY Rule)**: Your program must know its audience. If `stdout` is an interactive terminal, give me colors, spinners, and human-readable tables. If it's being piped to a file or another command, strip the ANSI codes and animations immediately. A progress bar in a CI log isn't a feature; it's a corrupt log file.
- **The "Help" Hierarchy**: Documentation isn't an afterthought. CLIG suggests a tiered approach: running the command bare should show concise usage examples (like `jq`), while `--help` should provide the full manual. Crucially, lead with **examples**, not syntax definitions. Users learn by pattern-matching, not by reading Backus–Naur form.
- **Flags over Arguments**: Positional arguments are brittle relics. While `cp source dest` is standard, it is also ambiguous. Modern tools should prefer flags (`--file`, `--output`) which are explicit, order-independent, and extensible without breaking backward compatibility.
- **JSON as a First-Class Citizen**: Since we operate in a web-centric world, tools should support a `--json` flag. This allows users to bypass text parsing tools like `awk` and pipe structured data directly into `jq` or networked services.

Why this matters

I value this guide because it treats CLI design with the same rigor we apply to graphical interfaces. It recognizes that **robustness** is a subjective feeling—a tool that catches interrupts (`Ctrl-C`) gracefully and provides feedback in under 100ms feels "solid," regardless of its actual execution time. A well-designed CLI respects my time by being discoverable and respects my mental model by being consistent.

Contributor: Alessandro Linzi