The Modern CLI Proficiency Examination: Who Are You in the Age of Advanced Tooling?

Preamble: Beyond the Shell - Assessing Your Command-Line Identity

Introduction

The command-line interface (CLI) has long been the definitive environment for the power user, a realm where efficiency and control are paramount. For decades, proficiency was measured by one's mastery of the POSIX standards—a canonical set of tools like ls, grep, cat, and sed that formed the bedrock of Unix-like systems. However, the landscape of software development and system administration has undergone a seismic shift. The complexity of modern workflows, the scale of data, and the interconnectedness of services demand more than what these venerable utilities can offer. In response, a new and vibrant ecosystem of advanced CLI tools has emerged.

This new generation of tooling is not merely an incremental improvement; it represents a philosophical evolution. Built with modern languages like Rust and Go, these tools prioritize performance, superior user experience (UX), and deep integration with developer workflows (e.g., Git, Docker, cloud services). They introduce interactive user interfaces (TUIs) that bring visual clarity to complex data, automate tedious command sequences, and reduce cognitive overhead. A developer's command-line identity is no longer defined solely by their knowledge of arcane sed incantations but by their fluency with this sophisticated, rapidly evolving toolkit.

This document serves as both a comprehensive examination and an educational treatise on the modern CLI. It is designed to benchmark your knowledge against the current state of the art, challenging you to look beyond the basics and assess your familiarity with the tools that define contemporary efficiency.

Methodology

The examination is structured into thematic parts, each focusing on a critical domain of CLI usage. Each subsection presents a question designed to test knowledge of a specific tool or concept. The questions are formatted with multiple-choice options for a clear assessment. Following each question is a detailed **Elucidation**. This section serves as the core of the report. It will not only provide the correct answer but will also offer a comprehensive analysis of the tool in question. This includes its key features, design philosophy, and a comparative analysis against its classic and modern alternatives, with direct support from documented evidence. The purpose of the elucidation is to provide a deep, contextual understanding that transforms this examination from a simple test into a valuable learning resource. By engaging with this material, you will not only measure your current proficiency but also gain a deeper appreciation for the power and elegance of the modern command line.

Part I: The New Guard - Modern Replacements for Core Utilities

This section assesses knowledge of the new generation of tools designed to replace or augment the classic Unix/Linux command-line utilities. The focus is on significant advancements in performance, user experience, and expanded feature sets that address the limitations of their predecessors in the context of modern development workflows.

Subsection 1.1: File Listing and Navigation

Question:
Which Rust-based ls replacement is specifically lauded for its rich feature set including grid-view, tree-view, and deep Git integration, and has recently been forked under the name eza due to the original project becoming unmaintained?
A) lsd
B) exa
C) tre
D) alder

Elucidation:
The correct answer is **B) exa**.
The classic ls command, while fundamental, provides information in a format that has remained largely unchanged for decades. In a modern development context where version control status and file type differentiation are critical, its output can feel spartan. This has led to the development of several "modern replacements" that aim to provide more context directly in the file listing.
exa emerged as a leading alternative, written in the Rust programming language.[1] Its

popularity stems from a rich feature set designed for developers. Key features include:

- **Enhanced Color Coding:** exa uses colors to distinguish between file types, permissions, and metadata far more effectively than the basic LS_COLORS implementation.
- **Grid and Tree Views:** It can display files in a standard grid but also includes a built-in tree view (--tree), replacing the need for a separate tree command.[1]
- **Git Integration:** Crucially, exa can query the Git index and display the version control status of each file directly in its output, indicating whether a file is new, modified, or staged.[2]
- **Extended Attributes:** It has built-in knowledge of file metadata, such as extended attributes and filesystem block sizes.

The development of tools like exa, lsd (ls deluxe), and tre signifies a broader trend in the CLI ecosystem: the "Rustification" of core utilities. Developers are choosing Rust for these projects for several compelling reasons. First, Rust provides memory safety guarantees without a garbage collector, which prevents common security vulnerabilities like buffer overflows that can plague C-based programs, while maintaining performance comparable to or exceeding that of C/C++. Second, this performance is a key selling point; these tools are designed to be fast, even with their enhanced feature sets.

This shift is not merely about the implementation language; it reflects a deeper change in design philosophy toward developer ergonomics. Features that were once considered luxuries or required complex shell scripting—such as Git integration and advanced color schemes—are now standard, built-in capabilities. The goal is to reduce the user's cognitive load by presenting rich, contextual information in a single, intuitive command. Recently, as the original exa project saw a slowdown in maintenance, the community forked it into a new project named eza to continue its development, which retains all the features of exa and continues to be actively maintained.[2]

Other alternatives mentioned, such as lsd and tre, also offer improvements over ls, with lsd being noted for its inclusion of icons and tre for its git awareness and editor aliasing.[2]

However,

exa/eza is the tool that most comprehensively matches the description in the question.

Subsection 1.2: File Content Viewing and Manipulation

Question:

When viewing a source code file in the terminal, which cat clone automatically provides syntax highlighting, Git gutter marks for modifications, and integrates with pagers like less by default?

A) most

B) highlight

C) bat

D) pygmentize

Elucidation:
The correct answer is **C) bat**.
The cat command (short for "concatenate") is traditionally used to print the contents of a file to standard output. While effective for plain text, it is code-agnostic. Viewing a source code file with cat results in a monolithic, uncolored block of text.
bat, self-described as "a cat(1) clone with wings," directly addresses this limitation.[1] Written in Rust, it serves the same primary purpose as
cat but with significant enhancements tailored for developers:

- **Syntax Highlighting:** bat automatically detects the programming language of a file and applies syntax highlighting, making code far more readable in the terminal.[1]
- **Git Integration:** It integrates with Git to display modifications in the "gutter" (the left margin). Lines that have been added, modified, or deleted since the last commit are marked with symbols, providing immediate visual context about the state of the file.[1]
- **Automatic Paging:** If a file's content is too large to fit on one screen, bat automatically pipes its output into a pager like less. This is a major usability improvement over cat, which would simply dump the entire file to the terminal, forcing the user to scroll back.
- **Non-printable Character Display:** It can show and highlight non-printable characters, which is useful for debugging formatting issues.

While other tools can achieve parts of this functionality (e.g., pygmentize is a library and command-line tool for syntax highlighting), bat is the tool that integrates all these features—syntax highlighting, Git integration, and automatic paging—into a single, cohesive package designed as a direct replacement for cat.
For file content manipulation, a similar modernization is occurring. The sed (stream editor) command is exceptionally powerful but is infamous for its cryptic and often unintuitive syntax. For common find-and-replace tasks, a tool like sd has gained popularity. It offers a more intuitive syntax (sd 'find_pattern' 'replace_string') that is easier to read and write than its sed equivalent (sed 's/find_pattern/replace_string/g').[1] Similarly, for comparing files,
diff-so-fancy builds upon the standard diff output to provide a more human-readable, character-level highlighting of changes, making it easier to spot subtle differences.[1] These tools, like
bat, exemplify a focus on improving the user experience for common developer tasks.

Subsection 1.3: Searching and Finding

Question:
You need to recursively search for a string pattern in a large Git repository. Which tool would be the most performant choice that automatically ignores files listed in .gitignore, hidden files, and binary files by default?
A) grep -r
B) ack

C) fzf
D) ripgrep (rg)

Elucidation:
The correct answer is **D) ripgrep (rg)**.
Searching for code is one of the most frequent operations in a developer's workflow. While grep is the classic tool, its performance can degrade in large directories, and it lacks awareness of version control contexts. This led to the creation of tools like ack and The Silver Searcher (ag), which introduced features like respecting ignore files.
ripgrep (rg), however, has largely superseded these earlier tools and is now widely considered the state of thethe-art for line-oriented code search.[1] Its advantages are numerous and significant:

- **Performance:** ripgrep is exceptionally fast. This speed is a direct result of its implementation. It is built on Rust's regex engine, which uses finite automata, SIMD (Single Instruction, Multiple Data) optimizations, and aggressive literal searching. Furthermore, it employs a lock-free parallel recursive directory iterator to search multiple files at once.[4]
- **Smart Filtering by Default:** Out of the box, ripgrep automatically respects rules in .gitignore, .hgignore, and .ignore files. It also skips hidden files/directories and binary files by default. This is a massive improvement over grep, which would require complex --exclude-dir and --exclude flags to replicate this behavior. This ensures search results are focused only on relevant source code.[4]
- **Unicode Support:** ripgrep handles Unicode correctly and performantly by default, building UTF-8 decoding directly into its automaton engine. This contrasts with GNU grep, which can suffer a significant performance penalty when handling Unicode.[4]
- **Feature Parity and Beyond:** It supports most of grep's features, such as context control (-C), file type filtering (-tpy for Python), and inverse matching (-v), while adding its own, like support for searching compressed files (-z) and using the PCRE2 regex engine for advanced features like look-around and backreferences (-P).[4]

For finding files by name, rather than content, fd is the modern counterpart to the classic find command. It offers a simpler, more intuitive syntax, is significantly faster, and also respects .gitignore rules by default, making it a more convenient choice in most development scenarios.[3]
A crucial aspect of the modern CLI is not just the power of individual tools, but their ability to be composed into interactive workflows. This is where fzf (fuzzy finder) becomes indispensable.[8]
fzf is an interactive filter that can take any list of items piped to it and allow the user to fuzzy-search and select from that list. The true power of the modern search toolkit is realized through the synergy of these components. For example:

1. A user can run fd. | fzf to get an interactive, fuzzy-searchable list of all files in a project.
2. A user can run rg "pattern" | fzf to interactively filter the results of a code search.
3. This composition can be extended, for instance, to preview files selected in fzf: fd. | fzf

--preview 'bat --color=always {}'.

Mastery of the modern CLI, therefore, is not just about knowing that rg is a fast grep. It is about understanding how to combine a fast file finder (fd), a fast content searcher (rg), and a powerful interactive filter (fzf) into fluid, efficient pipelines that solve complex problems with minimal keystrokes.

Table 1.1: Core Utility Replacements - A Comparative Analysis

Classic Command	Modern Replacement(s)	Key Advantages	Implementation Language
ls, tree	eza (formerly exa), lsd	Grid/tree views, Git integration, enhanced colors, icons.	Rust
cat	bat	Syntax highlighting, Git integration, automatic paging.	Rust
grep	ripgrep (rg)	Extreme performance, respects .gitignore by default, Unicode support.	Rust
find	fd	Simplified syntax, faster performance, respects .gitignore by default.	Rust
sed	sd	More intuitive syntax for find-and-replace operations.	Rust
diff	diff-so-fancy	More readable, character-level highlighting of differences.	Perl / Shell
cd	zoxide, autojump	Learns frequent directories for rapid navigation (z myproject).	Rust, Python

---

Part II: The Interactive Terminal - Mastering TUI-Based Workflows

This part delves into Terminal User Interfaces (TUIs), which represent a significant evolution in command-line interaction. By providing visual, interactive, and often keyboard-driven

interfaces, TUIs offer a powerful middle ground between raw command-line flags and full-fledged graphical applications. They excel at managing complex systems like databases, container runtimes, and version control, all without leaving the comfort and efficiency of the terminal.

Subsection 2.1: Database Management

Question:
The creator of the popular database TUIs pgcli and mycli was directly inspired by the user-friendly features of a REPL for which programming language, aiming to replicate its "pampering" experience of auto-completion and syntax-highlighting for SQL clients?
A) Ruby (IRB)
B) JavaScript (Node.js REPL)
C) Python (BPython)
D) Lisp (SLIME)

Elucidation:
The correct answer is **C) Python (BPython)**.
The default command-line clients for databases like PostgreSQL (psql) and MySQL (mysql) are powerful but lack modern interactive features. They typically do not offer auto-completion for table names, column names, or SQL keywords, nor do they provide syntax highlighting. This requires users to either have perfect recall of their database schema or constantly switch contexts to look it up.
The dbcli suite of tools was created to solve this problem. pgcli is a TUI for PostgreSQL [9], and mycli is its counterpart for MySQL, MariaDB, and Percona.[11] Both tools provide a vastly superior interactive experience with features such as:

- **Smart Auto-Completion:** As you type, they suggest context-sensitive completions. For example, after FROM, they will suggest table names; after WHERE, they will suggest column names from the relevant tables.[9] This feature can even complete aliases.[12]
- **Syntax Highlighting:** SQL queries are highlighted in real-time, improving readability and making it easier to spot syntax errors.[9]
- **Configuration and Favorites:** Users can save favorite queries as snippets and invoke them with special commands (e.g., \f <name>).[12]
- **Flexible Output Formatting:** Results can be displayed in various formats, including a vertical layout (\G) which is useful for wide tables.[12]

The genesis of these tools is a clear example of cross-ecosystem pollination of user experience concepts. The creator of pgcli explicitly stated that the inspiration came from BPython, an enhanced interactive shell for the Python programming language.[13] Having experienced the "pampering" of
BPython's excellent auto-completion and real-time feedback, the developer sought to bring that same level of interactive convenience to the domain of database management. This

demonstrates that innovative ideas in developer tooling are not confined to silos; a paradigm that improves the workflow in one area (a programming language REPL) can be successfully adapted to revolutionize another (a database client).
A similar philosophy is applied by iredis, a TUI for the Redis data store. It provides advanced auto-completion based on command history and even the results of previous commands (e.g., auto-completing a key for DEL based on the output of a KEYS command). It also includes a peek command that intelligently inspects a key's type and uses the appropriate command (GET, LRANGE, SSCAN, etc.) to display its value, saving the user multiple steps.[14]

Subsection 2.2: Docker and Container Management

Question:
You need a TUI to manage your Docker and docker-compose environments. You prefer a tool written in Go that is known for its minimal interface, colorful dashboard, and single-keystroke commands for common actions like viewing logs (m) or executing a shell (E). Which tool should you choose?
A) dockly
B) ctop
C) lazydocker
D) k9s

Elucidation:
The correct answer is **C) lazydocker**.
While the docker and docker-compose CLIs are the fundamental tools for container management, their commands can be verbose and require memorizing container IDs or names. For routine tasks like checking logs, restarting services, or shelling into a container, this can become tedious. TUIs for Docker aim to streamline these common workflows. lazydocker is a highly popular TUI for Docker management, written in Go.[2] It presents a multi-panel dashboard that provides an at-a-glance overview of containers, services, images, volumes, and networks.[16] Its primary appeal lies in its efficiency and ease of use, guided by a philosophy of "laziness"—that is, achieving results with the minimum possible effort. Key features include:

- **Single-Keystroke Shortcuts:** Nearly every common action is mapped to a single key. For example, pressing m on a selected container brings up its logs, E executes a shell inside it, s stops it, and r restarts it.[17]
- **Real-time Monitoring:** The interface displays real-time logs and resource usage (CPU/memory) for running containers.[16]
- **Bulk Operations:** It simplifies management of docker-compose stacks, allowing for bulk actions like bringing all services down, including volumes and images, with a single command (b for bulk, then selecting the desired action).[17]
- **Contextual Help:** A help menu (?) is always available, showing the possible actions for

the currently selected panel or item.[17]

Another prominent tool in this space is dockly, which is a Node.js application.[18] It provides similar functionality, offering an immersive dashboard for inspecting and managing containers and services, including swarm mode support.[18] While both tools solve the same core problem, lazydocker is often favored by those who prefer a Go-based, single-binary application and its specific keybinding scheme.

The emergence and popularity of tools like lazydocker and dockly (and k9s for Kubernetes [2]) highlight a significant trend. As the underlying systems (container runtimes, orchestrators) become more powerful and complex, the need for a higher-level interactive layer grows. The "lazy" philosophy is not about sloth but about efficiency. It recognizes that developer productivity is hampered by the cognitive friction of recalling and typing long, complex commands for frequent operations. These TUIs act as powerful abstractions, allowing users to manage complexity visually and interactively, directly within the terminal. It is important to distinguish these tools from similarly named but unrelated commercial products, such as Dockly, a retail operating system platform.[20]

Subsection 2.3: Git and Version Control

Question:

Which keyboard-driven Git TUI, praised for its speed and comprehensive feature set, allows a user to stage/unstage individual lines of a file, easily amend commits, and create pull requests, all without leaving the terminal?

A) tig
B) git-stats
C) grv
D) lazygit

Elucidation:

The correct answer is **D) lazygit**.

Git is an exceptionally powerful version control system, but its power comes with a steep learning curve and a command-line interface that can be complex for advanced operations. Tasks like interactive rebasing, staging specific lines (patching), or fixing up previous commits often involve multi-step commands and flags that can be difficult to remember.

lazygit is a terminal UI for Git, written in Go, that aims to make these complex operations simple and fast.[1] It provides a multi-panel interface showing the repository status, files, branches, commits, and stash.[21] Its killer feature is that it is completely keyboard-driven, allowing for a workflow that is much faster than traditional GUI clients like Sourcetree or GitKraken for users comfortable in the terminal.[23]

Key capabilities of lazygit that address common Git pain points include:

- **Interactive Staging:** A user can navigate to the "Files" panel, select a modified file, and press enter to view the diff. From there, individual lines or hunks can be staged or

unstaged with the spacebar, making it trivial to craft precise commits.[22]

- **Simplified Commit Actions:** Committing is a single keystroke (c). Amending the previous commit is also a single keypress. More advanced actions like squashing, fixing up, or editing commits are handled through simple interactive menus in the "Commits" panel.
- **Effortless Branch Management:** Switching branches, creating new branches from a commit, and performing actions like fast-forwarding or rebasing are all handled with simple keybindings in the "Branches" panel.[22]
- **Integration with Pull Requests:** After pushing a branch (P), lazygit can directly open the "create pull request" page in a web browser (o).[22]

tig is another well-regarded text-mode interface for Git, but its focus is more on browsing the repository history, viewing commits, and staging, acting as an interactive pager for Git.[3] While powerful,
lazygit is generally considered more comprehensive for performing a wider range of manipulative Git actions.
Like lazydocker, lazygit embodies the "lazy" philosophy of optimizing for efficiency by reducing cognitive load. It abstracts away the need to memorize the exact syntax for complex Git commands, replacing it with a more intuitive, visual, and interactive model. This allows developers to leverage the full power of Git with greater speed and less friction, keeping them in the "flow" of their work within the terminal.

Subsection 2.4: File Management

Question:
You are looking for a terminal file manager. You prioritize speed, a minimal memory footprint, and prefer configuration via environment variables over a dedicated config file. Which of the following, written in C, best fits your requirements?
A) ranger
B) nnn
C) lf
D) Vifm

Elucidation:
The correct answer is **B) nnn**.
While core utilities are sufficient for basic file operations, terminal file managers provide a more integrated and efficient experience for navigating and manipulating the filesystem. They offer a visual representation of the directory structure and unified keybindings for actions like copying, moving, deleting, and previewing files.
The TUI file manager space is characterized by a clear philosophical divide, best exemplified by the comparison between ranger and nnn.
ranger is a highly popular and feature-rich file manager written in Python.[24] It is known for:

- **Miller Columns:** A multi-column display that shows the directory hierarchy, similar to macOS Finder.[25]
- **VI-like Keybindings:** Navigation and commands are heavily inspired by the Vim text editor.[24]
- **Extensibility and Customization:** It is deeply configurable through dedicated files (rc.conf, commands.py) and comes with its own file opener, rifle, for setting up file associations.[24]
- **File Previews:** It can automatically preview the contents of the selected file, including text, images (if the terminal supports it), and more.[25]

nnn (n^3), on the other hand, represents the minimalist philosophy.[26] It is written in C with a strong focus on performance and resource efficiency, making it extremely fast and lightweight (typically using less than 3.5MB of memory).[28] Its distinguishing characteristics are:

- **Minimalism and Speed:** It is designed to be frugal and fast, with few library dependencies.[28]
- **Configuration via Environment Variables:** Unlike ranger, nnn does not use a configuration file. It is configured almost entirely through environment variables (e.g., NNN_BMS for bookmarks, NNN_PLUG for plugins), making it highly portable and consistent across different machines.[27]
- **Plugin Architecture:** While the core is minimal, functionality can be extended via a simple plugin system where users can add their own scripts.[27]
- **Contexts:** It features four "contexts" (tabs) to work in multiple directories simultaneously.[26]

Therefore, for a user who prioritizes raw speed, a low memory footprint, and a portable, file-less configuration method, nnn is the ideal choice. Other file managers like lf (written in Go) and Vifm (also C-based with vi-like keybindings) offer different balances of features and performance, but the ranger vs. nnn comparison most clearly illustrates the trade-off between feature-rich extensibility and minimalist performance.

Table 2.1: TUI Application Matrix - Functionality vs. Tool

Tool	Primary Function	Language	Key Feature 1	Key Feature 2	Configuration Style
lazygit	Git Version Control	Go	Single-key commands for complex actions	Interactive staging of individual lines	YAML config file
tig	Git Repository Browser	C	Text-mode interface for browsing history	Staging and commit viewing	~/.tigrc file
lazydocker	Docker/Compose Mgmt	Go	Single-key commands for	Real-time log and resource	YAML config file

			container actions	monitoring	
dockly	Docker/Compose Mgmt	Node.js	Immersive dashboard for containers/services	Docker Swarm support	N/A (CLI flags)
k9s	Kubernetes Mgmt	Go	Interactive dashboard for cluster resources	"X-Ray" and "Pulse" views for diagnostics	YAML config file
ranger	File Management	Python	Miller columns display	Highly extensible with rifle file opener	rc.conf, commands.py
nnn	File Management	C	Extremely fast and lightweight	Configuration via environment variables	Environment Variables
mycli	MySQL/MariaDB Client	Python	Context-aware auto-completion	Syntax highlighting, query snippets	~/.myclirc file
pgcli	PostgreSQL Client	Python	Context-aware auto-completion	Syntax highlighting, query snippets	~/.pgclirc file

---

Part III: The Developer's Arsenal - Code, Data, and DevOps

This section covers the specialized tools that are indispensable for the modern software development lifecycle. Moving beyond general-purpose utilities, these commands are tailored for specific, high-frequency tasks such as interacting with web APIs, manipulating structured data formats like JSON, and managing complex DevOps and cloud infrastructure workflows.

Subsection 3.1: Interacting with APIs and HTTP

Question:
Which command-line HTTP client is known for its human-friendly syntax, automatic JSON formatting, colorized output, and is often described as a "cURL for humans"?
A) wget
B) httpie

C) aria2
D) curlie

Elucidation:
The correct answer is **B) httpie**.
cURL is the de facto standard for making HTTP requests from the command line. It is incredibly powerful, versatile, and ubiquitous. However, its syntax can be verbose for common API interactions, and its output is raw by default, requiring piping to other tools like jq for formatting.
httpie was created to provide a more user-friendly experience specifically for interacting with web services and APIs.[1] Its design philosophy prioritizes ease of use and sensible defaults for the most common use cases. Its key advantages over
cURL include:

- **Simplified Syntax:** The command structure is more intuitive. For example, sending a POST request with JSON data is as simple as http POST example.com hello=world, whereas cURL would require curl -X POST -H "Content-Type: application/json" -d '{"hello": "world"}' example.com.
- **Automatic JSON Formatting:** httpie detects JSON responses and automatically formats and colorizes them for readability, eliminating the need to pipe the output to a separate tool.[2]
- **Colorized Output:** Both request and response components, including headers and status codes, are colorized to make them easier to parse visually.
- **Sensible Defaults:** It assumes common intentions, such as setting the Content-Type header to application/json when sending JSON data.

curlie is another modern alternative that aims to provide the power of cURL with the ease of use of httpie.[3] It acts as a frontend to
cURL, meaning it uses cURL's library (libcurl) under the hood and supports its arguments, but provides a friendlier interface. wget is primarily a file downloader, not a general-purpose HTTP client for API interaction. aria2 is a powerful download utility that supports multiple protocols, including HTTP, FTP, and BitTorrent, but is also not designed for interactive API testing.[1]
httpie is the tool that most famously carries the moniker "cURL for humans" and best matches the features described.

Subsection 3.2: Manipulating Structured Data (JSON, YAML)

Question:
You have a large, nested JSON response from an API and want to make it searchable with standard line-oriented tools like grep. Which utility would you use to transform the JSON into a flat, line-based format where each line represents a unique path to a value?
A) jq
B) fx

C) gron
D) jless

Elucidation:
The correct answer is **C) gron**.
JSON has become the lingua franca for APIs, but its nested, multi-line structure makes it difficult to process with traditional Unix tools like grep, sed, and awk, which are designed to operate on lines of text.
While jq is the undisputed champion for slicing, dicing, and transforming JSON with its own powerful query language [1], it has a learning curve. For the specific task of making JSON compatible with standard line-oriented tools,
gron ("grep-able JSON") provides an ingenious solution.[2]
gron takes a JSON object and "flattens" it into a series of simple assignment statements. Each line represents a unique path from the root of the object to a value.
For example, the JSON {"user": {"name": "Alice", "active": true}} would be transformed by gron into:

```
json.user.active = true;
json.user.name = "Alice";
```

This output is now trivial to process with grep. For instance, gron data.json | grep 'user.name' will reliably extract the line containing the user's name. gron can also perform the reverse operation, reconstructing a JSON object from the flattened format.
The ecosystem for handling structured data in the terminal is rich and growing. While gron excels at making JSON greppable, other tools address different needs:

- jq remains the most powerful tool for complex queries and transformations.[1]
- fx, jless, and jnv are interactive TUI viewers for JSON, allowing users to collapse and expand nodes and navigate large documents easily.[2]
- jqp provides a TUI playground for developing and testing jq filters interactively.[3]
- For YAML, yq serves a similar role to jq, providing a command-line processor for querying and manipulating YAML files.[2]

These tools demonstrate a clear understanding that while structured data is powerful, specialized utilities are needed to bridge the gap between its format and the line-oriented philosophy of the traditional command line.

Subsection 3.3: DevOps and Infrastructure Management

Question:
For managing Kubernetes resources, which TUI provides an interactive dashboard to view

pods, logs, and services, allowing for quick navigation and management of a cluster without writing full kubectl commands?
A) SAWS
B) lazydocker
C) k9s
D) htconvert

Elucidation:
The correct answer is **C) k9s**.
Managing modern infrastructure, particularly Kubernetes clusters, involves interacting with a complex, distributed system. The standard tool, kubectl, is powerful but can be very verbose. Simple tasks like checking the logs of a pod in a specific namespace can require long commands with multiple flags.
k9s is a Go-based TUI that provides a full-screen, interactive dashboard for Kubernetes clusters, dramatically simplifying cluster management.[2] It allows operators and developers to navigate and manage their clusters in a much more efficient and intuitive way.
k9s has become an indispensable tool for many Kubernetes practitioners due to features like:

- **Resource Navigation:** Users can quickly switch between viewing pods, services, deployments, configmaps, and other Kubernetes resources using simple commands (e.g., :pods, :svc).
- **Real-time Status:** The dashboard provides a real-time, color-coded view of the status of all resources, making it easy to spot pods that are crashing or pending.
- **Quick Actions:** Common actions are mapped to keystrokes. A user can view logs (l), shell into a pod (s), describe a resource (d), or delete a pod (ctrl-d) with minimal effort.
- **Diagnostic Tools:** k9s includes powerful diagnostic views like "Pulse" and "X-Ray" that provide a consolidated overview of the health and dependencies of cluster resources.
- **Plugin System:** Its functionality can be extended with custom plugins to perform specific actions.

k9s is to Kubernetes what lazydocker is to Docker—an interactive layer that reduces the cognitive load and verbosity of the underlying CLI, thereby increasing productivity. While lazydocker focuses on local container management [16],
k9s is designed for the distributed, multi-resource environment of Kubernetes. Other DevOps tools mentioned serve different purposes: SAWS is a supercharged CLI for AWS [2], and htconvert is a utility for converting .htaccess files to Nginx configurations.[2]
k9s is the definitive TUI for interactive Kubernetes management.

Part IV: The Organized Mind - Productivity from the Command Line

This part examines a category of tools that bring structured productivity methodologies into

the terminal. For users who live in the command line, switching to a graphical application for task management, time tracking, or note-taking can be a disruptive context switch. These tools allow for the seamless integration of personal and professional organization into a developer's primary work environment.

Subsection 4.1: Task and To-Do List Management

Question:
Which command-line task management tool is known for its flexible, methodology-neutral approach, use of a .taskrc configuration file, and its ability to handle complex queries like task project:Home +weekend priority:H list?
A) doing
B) todo.txt
C) Taskwarrior
D) taskbook

Elucidation:
The correct answer is **C) Taskwarrior**.
Taskwarrior is a sophisticated, open-source task management system that operates entirely from the command line.[29] It is designed to be fast, flexible, and unobtrusive, allowing a user to capture a task and immediately return to their work.
While it can be used for simple to-do lists with just three core commands (add, list, done), its true power lies in its rich feature set and data model.[29] Key features include:

- **Rich Metadata:** Tasks are not just descriptions. They can have attributes like a project, priority, due date, recurrence rules, and arbitrary tags.[31]
- **Powerful Filtering:** Taskwarrior has an expressive query language that allows for complex filtering of tasks. The example in the question, task project:Home +weekend priority:H list, demonstrates filtering by project, tag, and priority simultaneously.[32]
- **Methodology-Neutral:** While inspired by concepts from Getting Things Done (GTD), Taskwarrior does not enforce any particular productivity methodology. It provides the tools for users to implement the system that works best for them.[29]
- **Extensibility:** It is highly configurable via the ~/.taskrc file and supports hook scripts that can trigger custom actions on events like adding or modifying a task. This allows for deep integration with other scripts and workflows.[33]
- **Synchronization:** In conjunction with Taskserver, Taskwarrior data can be synced across multiple machines, providing a robust, self-hostable solution for task management.[31]

Other tools like taskbook and doing provide simpler, more focused to-do list functionality.[1] The
todo.txt format is a well-known convention for managing tasks in a plain text file. However, Taskwarrior stands out for its database-like approach to task management, its powerful query

engine, and its deep customizability, making it the premier choice for users who require a sophisticated task management system within the terminal.

Subsection 4.2: Time Tracking

Question:
You use Taskwarrior for task management and want a companion tool from the same developers to track time spent on those tasks. Which tool is designed for this specific integration, allowing you to start and stop timers associated with your tasks and generate reports on time usage?
A) Watson
B) Timewarrior
C) moro
D) Timetrap

Elucidation:
The correct answer is **B) Timewarrior**.
Timewarrior is a command-line time tracking utility developed by the same team as Taskwarrior and is designed to be its official companion.[35] While
Taskwarrior focuses on what needs to be done (the future), Timewarrior focuses on accurately recording time that has been spent (the past and present).[31]
The key feature of Timewarrior is its seamless integration with Taskwarrior. When you start tracking a task in Taskwarrior, a hook script can automatically start a Timewarrior timer with the corresponding tags and project from the task. When the task is stopped or completed, the timer is also stopped.[35] This creates a powerful, unified system for managing both tasks and the time spent on them.
Core features of Timewarrior include:

- **Simple Interface:** The basic commands are start, stop, continue, and summary.
- **Tagging:** Time intervals can be tagged with arbitrary labels, allowing for detailed reporting.
- **Reporting:** It can generate summary reports and visual charts showing how time was spent over various periods (day, week, month).[35]
- **Data Correction:** It provides commands to correct mistakes, such as forgetting to stop a timer (shorten, lengthen, move).
- **Configuration:** It can be configured to understand your work week, holidays, and vacations, allowing it to generate more accurate reports about productive time.[35]

Watson is another very popular command-line time tracker written in Python.[2] It stores its data in a simple, human-readable JSON file and also supports projects and tags. It is an excellent standalone tool and can be integrated with other systems via scripting.[40] However, Timewarrior is the tool specifically designed and maintained by the Taskwarrior developers for direct, out-of-the-box integration. It is important not to confuse Watson the time tracker with

the IBM Watson AI platform, which is an entirely different suite of cloud-based AI services.[42] Other tools like
Timetrap and moro also offer command-line time tracking but lack the deep integration with Taskwarrior that Timewarrior provides.[2]

Subsection 4.3: Note-Taking and Knowledge Management

Question:
For users who wish to manage their Evernote notebooks from the terminal, which Python-based client allows for creating notes in Markdown, searching with filters, and editing notes using a configured console editor like vim?
A) sncli
B) geeknote
C) nb
D) obs

Elucidation:
The correct answer is **B) geeknote**.
While many knowledge management and note-taking platforms are GUI-centric, there is a persistent demand from some users to integrate these services into their terminal-based workflows. geeknote is a command-line client for Evernote that caters specifically to this need.[44] It allows users to perform most core Evernote functions without leaving the terminal. Key features of geeknote include:

- **Note Creation and Editing:** Users can create new notes directly from the command line. For editing, geeknote shells out to the user's configured text editor (e.g., vim, emacs, nano), allowing them to use a familiar, powerful editing environment. It supports writing notes in Markdown.[46]
- **Comprehensive Search:** It provides a find command with filters for searching notes by keywords, tags, notebooks, and dates.[44]
- **Notebook and Tag Management:** Users can create, rename, and delete notebooks and tags directly.[44]
- **Scriptability:** As a command-line tool, it can be easily integrated into shell scripts or cron jobs for automating note creation or synchronization.[44]
- **File Syncing:** It includes a utility called gnsync which can synchronize a local directory of text files with an Evernote notebook.[45]

This tool demonstrates the desire of CLI power users to maintain a consistent workflow, even for services that are not traditionally CLI-first. Other tools in this space target different platforms; for example, sncli is a client for Simplenote, and obs is a CLI for interacting with Obsidian vaults.[2]
geeknote is the specific tool designed for Evernote integration.

Part V: Bridging Worlds - Networking and External Service Integration

This section covers tools that are critical for modern development, acting as bridges between a developer's local machine and the wider internet. These utilities solve the fundamental problem of exposing local services for webhook testing, client demos, and collaborative development, bypassing the complexities of network configuration like NAT and firewalls.

Subsection 5.1: Tunneling and Exposing Localhost

Question:
You are developing an application that receives webhooks from a third-party service. You need to expose your local development server to a public URL and also inspect the incoming HTTP requests and headers for debugging. Which tool provides a built-in web interface for this traffic inspection and replay capability?
A) localtunnel
B) serve
C) ngrok
D) mosh

Elucidation:
The correct answer is **C) ngrok**.
A common challenge in modern web development is testing integrations with external services that use webhooks (e.g., Stripe, GitHub, Slack). These services need to send HTTP requests to a publicly accessible URL, but the developer's application is typically running on localhost, which is unreachable from the public internet. Tunneling services solve this by creating a secure tunnel from a public endpoint to the local machine.
ngrok is arguably the most well-known and feature-rich tool in this category.[48] It is more than just a simple tunnel; it is a sophisticated "ingress-as-a-service" platform. When you run ngrok http 8080, it provides a public URL (e.g., https://random-string.ngrok.io) that forwards all traffic to your local port 8080. Its standout features for developers include:

- **Traffic Inspection and Replay:** ngrok provides a local web interface (typically at http://localhost:4040) that displays every request and response that passes through the tunnel in real-time. This allows developers to inspect headers, payloads, and status codes. Crucially, it also allows requests to be "replayed" with a single click, which is invaluable for debugging webhooks without having to trigger the external event repeatedly.[48]
- **Secure Tunnels:** All tunnels are secured with HTTPS by default, handling SSL/TLS termination automatically.[48]

- **Protocol Support:** It supports not only HTTP/HTTPS but also raw TCP tunnels, which can be used for services like SSH or databases.[48]
- **Authentication and Security:** Paid plans offer advanced features like adding OAuth/SAML/OIDC for SSO, IP whitelisting, mutual TLS, and webhook signature verification directly at the edge, without requiring code changes in the local application.[51]

localtunnel is a popular open-source alternative.[53] It is a NodeJS-based tool that provides the same core functionality: assigning a unique public URL to a local port.[53] It is simpler, free, and excellent for quick sharing or simple tunneling needs. However, it does not have the built-in traffic inspection and replay GUI that is a hallmark of
ngrok.

The evolution of ngrok from a simple utility to a full-fledged commercial platform illustrates a significant trend: the "platformization" of CLI tools. The problem of providing reliable, secure, and observable global ingress is complex and requires significant infrastructure. This naturally leads to a commercial, service-based model rather than a simple, standalone open-source tool. A developer's choice between ngrok and localtunnel is therefore not just a technical one, but also a decision between a feature-rich, proprietary platform with a freemium model and a simpler, community-supported open-source tool.

Table 5.1: Tunneling Services - A Feature Comparison

Feature	ngrok	localtunnel
Setup Requirement	Single, self-contained binary	NodeJS and npm install -g localtunnel
Core Functionality	Creates secure HTTP/S and TCP tunnels to localhost.	Creates secure HTTP/S tunnels to localhost.
Traffic Inspection	Built-in web interface (localhost:4040) for inspecting and replaying all requests.	No built-in inspection GUI.
Custom Domains	Supported on paid plans (e.g., demo.mycompany.com).	Supports requesting a specific subdomain (e.g., mysubdomain.loca.lt).
Authentication	Advanced options on paid plans: OAuth, SAML, OIDC, mTLS, IP whitelisting.	No built-in authentication features.
Business Model	Freemium (commercial platform with free and paid tiers).	Free and Open Source.
Primary Use Case	Professional development, robust webhook testing, client demos, production ingress.	Quick, simple sharing of development work, basic tunneling needs.

Part VI: The Art of the Terminal - Shell Enhancement and Customization

This final section explores the tools and frameworks that transform the terminal from a purely functional utility into a personalized, aesthetically pleasing, and hyper-efficient workspace. This practice, often called "ricing," reflects a culture where the command line is not just a tool, but a craft and a form of self-expression for the developer.

Subsection 6.1: Shell Frameworks and Prompts

Question:
Which community-driven framework for Zsh provides a vast collection of plugins and themes out-of-the-box, and is configured by setting variables like ZSH_THEME and adding to the plugins array in the ~/.zshrc file?
A) prezto
B) bash-it
C) oh-my-zsh
D) starship

Elucidation:
The correct answer is **C) oh-my-zsh**.
Zsh (the Z shell) is a powerful shell with many advanced features over the more traditional Bash, such as improved tab completion, spelling correction, and globbing. However, configuring these features from scratch can be a complex and time-consuming task.
oh-my-zsh is an open-source, community-driven framework that makes managing a Zsh configuration incredibly easy.[55] It is not a shell itself, but a collection of scripts, functions, and configurations that enhance the Zsh experience. Its immense popularity is due to two main components:

- **Plugins:** oh-my-zsh comes bundled with over 300 optional plugins for a wide array of tools and languages (e.g., git, docker, python, node).[55] These plugins provide useful aliases, functions, and completion definitions. For example, the
  git plugin alone provides dozens of short aliases for common Git commands (gco for git checkout, gp for git push, etc.), significantly speeding up version control workflows.[57]
- **Themes:** It includes over 140 themes that customize the appearance of the shell prompt.[55] These themes can display useful, context-aware information, such as the current Git branch, its status (e.g., number of commits ahead/behind remote), the active Python virtual environment, and more.[58]

Configuration is handled simply by editing the ~/.zshrc file. A user enables a theme by setting ZSH_THEME="theme-name" and enables plugins by adding their names to an array, for

example: plugins=(git docker python). This simple, declarative approach abstracts away the complexity of manual Zsh configuration, making it accessible to a much wider audience. While some critics argue that it can be "bloated" due to the number of scripts it loads, its ease of use and the breadth of its community contributions have made it the de facto starting point for most new Zsh users.[60]

Subsection 6.2: Terminal Aesthetics and "Ricing"

Question:
You want to automate the theming of your terminal and other applications to match your desktop wallpaper. Which tool is specifically designed to generate a 16-color palette from an image and apply it system-wide using templates?
A) themer
B) pywal
C) base16-builder
D) lolcat

Elucidation:
The correct answer is **B) pywal**.
For many developers who spend their entire day in the terminal, personalizing that environment is a way to make it more comfortable and aesthetically pleasing. This practice of customizing the visual appearance of one's desktop and applications is known as "ricing." pywal is a Python tool at the heart of many ricing workflows.[61] Its core function is to take an image, analyze it to find the dominant colors, and generate a 16-color palette (8 base colors and 8 bright variants).[63] It then takes this palette and applies it on-the-fly to the system. Its mechanism is both simple and powerful:

- **Color Generation:** It uses one of several backends (the default being imagemagick) to extract a color palette from a given wallpaper image.[64]
- **Live Terminal Theming:** It uses special escape sequences to change the colors of all open terminal emulators in real-time.[64]
- **Template and Export System:** pywal exports the generated colors into various formats and caches them in ~/.cache/wal/. It can then use templates to apply these colors to a wide range of other applications, such as tiling window managers (i3, bspwm), status bars, text editors, and more.[64]
- **Persistence:** By adding a simple command to shell startup files (e.g., .bashrc) and X session startup files (e.g., .xinitrc), the theme can be made to persist across new terminal sessions and reboots.[64]

The existence and popularity of tools like pywal and frameworks like oh-my-zsh demonstrate that the relationship with the CLI is not always purely utilitarian. The terminal is a developer's primary workspace, and customizing it is a form of craftsmanship and personal expression. The choice of prompt, font, and color scheme is a way to create a unique, motivating, and

comfortable environment. This aesthetic dimension is a key part of what defines a user's modern command-line identity; it reflects how they choose to *see* and interact with their digital world. While a tool like lolcat can add rainbow colors to output, pywal is designed for creating a cohesive, system-wide theme.[1]

Conclusion: A Synthesis of Tools and Philosophy

This examination has traversed the landscape of the modern command-line interface, revealing a dynamic and rapidly evolving ecosystem. The journey from classic POSIX utilities to the current state of the art is not merely a story of new tools, but of new philosophies driving their creation. Several key trends have emerged that collectively define the identity of the contemporary CLI power user.

First is the profound impact of modern systems programming languages, particularly Rust. The "Rustification" of core utilities—seen in ripgrep, eza, bat, and fd—is a deliberate move toward tools that are not only faster but also safer and more memory-efficient than their C-based predecessors. This technical foundation enables a second, more user-facing trend: a relentless focus on developer ergonomics. Features like intelligent defaults, context-aware integrations with Git, and superior visual feedback are no longer afterthoughts but core design principles.

Third, the rise of the Terminal User Interface (TUI) marks a significant maturation of the CLI. Tools like lazygit, k9s, and mycli represent an acknowledgment that for managing complex systems, a purely flag-based interface creates excessive cognitive load. These TUIs provide a visual, interactive layer that streamlines common workflows, embodying a "lazy" philosophy where efficiency is prioritized over the rote memorization of commands. This paradigm shift was itself born from the cross-pollination of ideas, with the user-friendly experience of a Python REPL inspiring a revolution in database clients.

Fourth, the CLI has become a primary interface for consuming sophisticated, commercial developer platforms. The evolution of ngrok from a simple utility to a full-fledged "ingress-as-a-service" platform shows that as the problems being solved become more complex, the tools to solve them are increasingly backed by commercial infrastructure and business models. This introduces a new dimension to a developer's toolkit, blending open-source ideals with pragmatic platform integration.

Finally, the terminal has solidified its role not just as a workspace, but as a canvas for personal expression. The culture of "ricing," powered by tools like oh-my-zsh and pywal, underscores that a developer's relationship with their environment is deeply personal. The customization of prompts, colors, and workflows is a form of craftsmanship that makes the digital workspace both more efficient and more enjoyable.

Ultimately, the question "Who are you without the CLI?" is answered by the tools one chooses to wield. This choice reflects a personal philosophy: Are you a minimalist who values raw C-based speed (nnn)? An efficiency expert leveraging Go-based TUIs to manage complexity (lazygit, k9s)? A pragmatist integrating powerful, paid platforms into your workflow (ngrok)?

Or an artisan who crafts a beautiful and bespoke environment (pywal, oh-my-zsh)? The modern CLI is a vast and varied arsenal. True proficiency lies not only in knowing individual commands but in understanding the philosophies they represent and composing them into a workflow that is a true extension of one's own identity as a builder and a problem-solver. The future of the command line points toward an even more powerful, interactive, and intelligent environment, further blurring the lines between text, interface, and service, and continuing to be the definitive domain of the power user.

Источники

1. CLI tools you won't be able to live without - DEV Community, дата последнего обращения: августа 8, 2025, https://dev.to/lissy93/cli-tools-you-cant-live-without-57f6
2. agarrharr/awesome-cli-apps: A curated list of command line ... - GitHub, дата последнего обращения: августа 8, 2025, https://github.com/agarrharr/awesome-cli-apps
3. My List of CLI Gems - Dolev Hadar, дата последнего обращения: августа 8, 2025, https://www.dlvhdr.me/posts/cli-tools
4. ripgrep recursively searches directories for a regex pattern while respecting your gitignore - GitHub, дата последнего обращения: августа 8, 2025, https://github.com/BurntSushi/ripgrep
5. ripgrep 0.7.1 - Docs.rs, дата последнего обращения: августа 8, 2025, https://docs.rs/ripgrep/0.7.1
6. How to Use the Fast and Powerful ripgrep Command-Line Tool - Gcore, дата последнего обращения: августа 8, 2025, https://gcore.com/learning/use-ripgrep-command-line-tool
7. List of CLI programs (follow-up to GUI). Feel free to make suggestions. : r/linux - Reddit, дата последнего обращения: августа 8, 2025, https://www.reddit.com/r/linux/comments/s8bbxq/list_of_cli_programs_followup_to_gui_feel_free_to/
8. fzf - junegunn.choi., дата последнего обращения: августа 8, 2025, https://junegunn.github.io/fzf/
9. A REPL for Postgres — pgcli latest documentation, дата последнего обращения: августа 8, 2025, https://pgcli.readthedocs.io/en/latest/
10. pgcli, дата последнего обращения: августа 8, 2025, https://www.pgcli.com/
11. mycli, дата последнего обращения: августа 8, 2025, https://www.mycli.net/
12. MySQL Command Line Tool: Unleash the Power of mycli - The Valuable Dev, дата последнего обращения: августа 8, 2025, https://thevaluable.dev/mysql-command-line-tool-mycli/
13. Faq - mycli, дата последнего обращения: августа 8, 2025, https://www.mycli.net/faq
14. IRedis | PitchWall, дата последнего обращения: августа 8, 2025, https://pitchwall.co/startup/iredis
15. IRedis · A Terminal Client for Redis with AutoCompletion and Syntax Highlighting, дата последнего обращения: августа 8, 2025, https://iredis.xbin.io/

16. What is lazydocker and how to use it. - NashTech Blog, дата последнего обращения: августа 8, 2025, https://blog.nashtechglobal.com/lazydocker-docker/
17. Get more productive: Introducing LazyDocker - Laurent Meyer's Devblog, дата последнего обращения: августа 8, 2025, https://meyer-laurent.com/get-more-productive-introducing-lazydocker
18. dockly - immersive terminal interface for managing docker containers and services, дата последнего обращения: августа 8, 2025, https://lirantal.github.io/dockly/
19. Lazydocker: a lazier way to manage everything Docker | Hacker News, дата последнего обращения: августа 8, 2025, https://news.ycombinator.com/item?id=36778905
20. Dockly, дата последнего обращения: августа 8, 2025, https://www.allmond.ai/
21. How to Use Lazygit to Improve Your Git Workflow - freeCodeCamp, дата последнего обращения: августа 8, 2025, https://www.freecodecamp.org/news/how-to-use-lazygit-to-improve-your-git-workflow/
22. lazygit deep dive - Introduction to a new series - oliverguenther.de, дата последнего обращения: августа 8, 2025, https://oliverguenther.de/2021/04/lazygit-an-introduction-series/
23. Level-up Git with lazygit. Whether you use git cli commands... | by Rasmus Hansen-Smith, дата последнего обращения: августа 8, 2025, https://medium.com/@rasmusfangel/level-up-git-with-lazygit-b5e6c923c5d7
24. ranger - ArchWiki, дата последнего обращения: августа 8, 2025, https://wiki.archlinux.org/title/Ranger
25. ranger (file manager) - Wikipedia, дата последнего обращения: августа 8, 2025, https://en.wikipedia.org/wiki/Ranger_(file_manager)
26. nnn (file manager) - Wikipedia, дата последнего обращения: августа 8, 2025, https://en.wikipedia.org/wiki/Nnn_(file_manager)
27. nnn - ArchWiki, дата последнего обращения: августа 8, 2025, https://wiki.archlinux.org/title/Nnn
28. jarun/nnn: n³ The unorthodox terminal file manager - GitHub, дата последнего обращения: августа 8, 2025, https://github.com/jarun/nnn
29. Taskwarrior - What's next?, дата последнего обращения: августа 8, 2025, https://taskwarrior.org/docs/start/
30. Taskwarrior, дата последнего обращения: августа 8, 2025, https://taskwarrior.org/
31. Taskwarrior - Wikipedia, дата последнего обращения: августа 8, 2025, https://en.wikipedia.org/wiki/Taskwarrior
32. Taskwarrior Review 2025 - Features, Pricing, Hacks and Tips - Productivity Directory, дата последнего обращения: августа 8, 2025, https://productivity.directory/taskwarrior
33. How to Request a Feature - Taskwarrior, дата последнего обращения: августа 8, 2025, https://taskwarrior.org/docs/features/
34. Taskwarrior - Documentation, дата последнего обращения: августа 8, 2025,

https://taskwarrior.org/docs/
35. Timewarrior 1.0.0 Released - Taskwarrior, дата последнего обращения: августа 8, 2025, https://taskwarrior.org/news/news.20160821/
36. Timewarrior, дата последнего обращения: августа 8, 2025, https://timewarrior.net/
37. Timewarrior 1.1.0 Released - Taskwarrior, дата последнего обращения: августа 8, 2025, https://taskwarrior.org/news/news.20180113/
38. timew-summary - Timewarrior, дата последнего обращения: августа 8, 2025, https://timewarrior.net/docs/summary/
39. Watson - GitHub Pages, дата последнего обращения: августа 8, 2025, https://jazzband.github.io/Watson/
40. Time and Truth: Lessons from Tracking My Work with Watson - DEV Community, дата последнего обращения: августа 8, 2025, https://dev.to/vst/time-and-truth-lessons-from-tracking-my-work-with-watson-anp
41. Time Tracking with Watson and Tmux - Elijah Manor, дата последнего обращения: августа 8, 2025, https://elijahmanor.com/blog/watson-tmux
42. Using the watsonx.ai Time Series Forecasting API to predict energy demand | IBM, дата последнего обращения: августа 8, 2025, https://www.ibm.com/think/tutorials/time-series-api-watsonx-ai
43. About watsonx Assistant - IBM Cloud Docs, дата последнего обращения: августа 8, 2025, https://cloud.ibm.com/docs/watson-assistant?topic=watson-assistant-about
44. vitalyrodnenko/geeknote: Console client for Evernote. - GitHub, дата последнего обращения: августа 8, 2025, https://github.com/vitalyrodnenko/geeknote
45. Geeknote: Command-Line Evernote Client ~ Web Upd8: Ubuntu / Linux blog, дата последнего обращения: августа 8, 2025, http://www.webupd8.org/2014/09/geeknote-command-line-evernote-client.html
46. Evernote layer - Spacemacs, дата последнего обращения: августа 8, 2025, https://www.spacemacs.org/layers/+web-services/evernote/README.html
47. 5 Interesting Third Party Evernote Clients For Linux & Mac - MakeUseOf, дата последнего обращения: августа 8, 2025, https://www.makeuseof.com/tag/5-interesting-third-party-evernote-clients-linux-mac/
48. What is Ngrok and How Does It Work? | BrowserStack, дата последнего обращения: августа 8, 2025, https://www.browserstack.com/guide/what-is-ngrok
49. ngrok | API Gateway, Kubernetes Ingress, Webhook Gateway, дата последнего обращения: августа 8, 2025, https://ngrok.com/
50. What is Ngrok? and 5 reasons why should you use it! - Muktar SayedSaleh | مختار سيد صالح, дата последнего обращения: августа 8, 2025, https://muktar.tech/what-is-ngrok-and-5-reasons-why-should-you-use-it-32474d17138c
51. Product - Ngrok, дата последнего обращения: августа 8, 2025, https://ngrok.com/our-product

52. Overview | ngrok documentation, дата последнего обращения: августа 8, 2025, https://ngrok.com/docs/
53. Localtunnel ~ Expose yourself to the world, дата последнего обращения: августа 8, 2025, https://localtunnel.me/
54. Local Tunnel - Upstash Documentation, дата последнего обращения: августа 8, 2025, https://upstash.com/docs/qstash/howto/local-tunnel
55. ohmyzsh/ohmyzsh: A delightful community-driven (with 2,400+ contributors) framework for managing your zsh configuration. Includes 300+ optional plugins (rails, git, macOS, hub, docker, homebrew, node, php, python, etc), 140+ themes to spice up your morning, and an - GitHub, дата последнего обращения: августа 8, 2025, https://github.com/ohmyzsh/ohmyzsh
56. Oh My Zsh - a delightful & open source framework for Zsh, дата последнего обращения: августа 8, 2025, https://ohmyz.sh/
57. My favourite Zsh features - JoeJag, дата последнего обращения: августа 8, 2025, https://code.joejag.com/2014/why-zsh.html
58. Why Oh My ZSH is so cool? - DEV Community, дата последнего обращения: августа 8, 2025, https://dev.to/equiman/why-oh-my-zsh-is-so-cool-31gd
59. Themes · ohmyzsh/ohmyzsh Wiki - GitHub, дата последнего обращения: августа 8, 2025, https://github.com/ohmyzsh/ohmyzsh/wiki/themes
60. Is Oh-My-Zsh worth it? - Reddit, дата последнего обращения: августа 8, 2025, https://www.reddit.com/r/zsh/comments/1iu8bz8/is_ohmyzsh_worth_it/
61. pywal | x-cmd pkg | Pywal is a tool that generates a color palette from the dominant colors in an image, дата последнего обращения: августа 8, 2025, https://www.x-cmd.com/pkg/pywal/
62. dylanaraps/pywal: Generate and change color-schemes on the fly. - GitHub, дата последнего обращения: августа 8, 2025, https://github.com/dylanaraps/pywal
63. Pywal related questions for Hyprland - Reddit, дата последнего обращения: августа 8, 2025, https://www.reddit.com/r/hyprland/comments/1ba19re/pywal_related_questions_for_hyprland/
64. dylanaraps/wal: Generate and change colorschemes on the fly. Deprecated, use pywal instead. - GitHub, дата последнего обращения: августа 8, 2025, https://github.com/dylanaraps/wal
65. Using python-pywal to make awesome screenshots - ArcoLinux, дата последнего обращения: августа 8, 2025, https://arcolinux.com/using-python-pywal-to-make-awesome-screenshots/