Network Working Group
Internet-Draft
Intended status: Informational
Expires: 16 September 2026

G. Fragkos
Independent

15 March 2026

A Standardized Set of ASCII-Based 3-Character Prefixes for Command-Line Interface Standard Output and Plaintext Logs to Aid in Software Development by Humans and AI-Generated Source Code

draft-fragkos-cliprefix-standard-00

Abstract

Command-line interface (CLI) tools frequently emit status, diagnostic, and progress messages that in some cases are prefixed with short ASCII tokens containing anything from any printable characters (e.g. !, i, +, etc.), abbreviations (e.g. DEL, INS, etc.), to short verbal descriptors (e.g. "INFO:", "DELETE", "ALERT", etc.). Despite their ubiquity, use, and convenience, no standard defines these tokens or their semantics. This document proposes a machine-parseable, human-readable vocabulary of 3-character ASCII prefixes -- collectively referred to as "gclifixes" -- enclosed in square brackets [ ] for events and curly braces { } for context metadata. The goal is to enable consistent interpretation across CLI tools, log aggregators, CI/CD pipelines, IDE integrations, and most importantly for any AI-generated code the adoption of this standard will streamline how these are used and provides additional meaning to any prompt message and log file without additional effort.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 16 September 2026.

Table of Contents

1. Introduction

1.1. Motivation and Prior Work

CLI tools across every operating system and programming ecosystem emit human-readable messages during execution. Developers, system administrators, and automated pipelines rely on these messages for monitoring, debugging, and auditing. A de facto convention has emerged whereby a short ASCII token in square brackets precedes the message text to indicate its category:

```
[+] User created successfully
[!] Certificate expires in 3 days
[x] FATAL: Cannot open database
```

Despite the widespread adoption of such pattern among other similar forms of visually contextualizing the displayed messages, no formal specification exists. The consequences include:

Inconsistent semantics: one tool uses [*] for errors while another uses it for progress. The same symbol carries different meanings across different tools, making log aggregation and automated parsing unreliable.

Parser fragility: log analysis tools, CI/CD dashboards, and IDE integrations must implement per-tool heuristics to interpret prefix meanings, leading to brittle integrations that break when tools update their output format.

AI code generation ambiguity: large language models generating CLI tool code have no authoritative reference for which prefix to use for which condition, resulting in inconsistent output conventions across generated code.

No straightforward human-readable log context: irrespective of the presence of logs, different messages can be interpreted in different ways. In larger implementations there are log entries that may have log IDs which are product specific and require significant effort to utilize. Such numeric log IDs can be extremely difficult to be read by humans or to parse fast for any context-related log messages. The use of 3-character standardized prefixes (gclifixes) provides the necessary context for each log entry, and enables extremely fast adoption.

Existing logging frameworks (syslog [RFC5424], Log4j, Python logging) define severity levels but not the broader vocabulary of operational semantics (data flow, process lifecycle, concurrency, security context) that CLI output routinely conveys. This specification fills that gap at the presentation layer, following the approach taken by [RFC9116] which similarly standardized a previously ad-hoc convention (security.txt) into a machine-parseable format.

1.2. Scope

This document defines a vocabulary of ASCII-based 3-character prefixes (gclifixes) for use in CLI output messages and structured log lines. It covers:

(a) The format of a prefix token (Section 2.1).

(b) A two-tier bracket system separating events from context (Section 2.2).

(c) A registry of assigned prefix characters and their semantics (Sections 3 and 4).

(d) An ABNF grammar for machine parsing (Section 5).

(e) Implementation guidance including colour mapping and AI integration (Section 7).

This specification does NOT define log transport protocols, structured logging wire formats (e.g., JSON, CBOR), or log storage schemas. It is a presentation-layer convention that can be adopted independently of the underlying infrastructure.

1.3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are used throughout this document:

gclifix (singular), gclifixes (plural): The informal name for the 3-character ASCII prefixes defined by this specification. Derived from "Generic CLI Prefix." The term is intended for use in conversation, documentation, and tooling (e.g., "add gclifixes to your log output", "parse for the [o] gclifix").

Prefix: A 3-character ASCII token consisting of an opening bracket, a single semantically meaningful character, and a closing bracket (e.g., [+] or {#}). Informally called a "gclifix."

Event Prefix (Event Gclifix): A prefix enclosed in square brackets [ ] that describes WHAT happened (an action, outcome, or state change).

Context Prefix (Context Gclifix): A prefix enclosed in curly braces { } that describes the CONDITIONS under which something happened (identity, scope, environment, temporal position).

Emitter: Any software component that produces prefixed output lines (CLI tools, logging libraries, CI/CD runners, etc.).

Consumer: Any software component or human operator that reads and interprets prefixed output lines (log viewers, dashboards, parsers, grep pipelines, etc.).

2. The Specification

2.1. Prefix Format

A gclifix token is exactly 3 characters long and conforms to one of two patterns:

```
Event  Prefix:   "[" <char> "]"
Context Prefix:  "{" <char> "}"
```

Where <char> is a single ASCII character from the assigned registry (Sections 3 and 4). The character MUST be drawn from the printable ASCII range (0x21-0x7E) excluding the bracket characters themselves.

A gclifix token MUST be followed by at least one space (0x20) before the next token or the message body. A line MUST contain at most one event gclifix. The event gclifix MUST appear as the first (leftmost) prefix on the line. A line MAY contain zero or more context gclifixes, all of which MUST follow the event gclifix.

2.2. Two-Tier Bracket System

This specification defines two complementary layers of meaning:

Square Brackets [ ] -- Event Tier: Describes WHAT happened. Every prefixed output line SHOULD contain exactly one event gclifix. The event gclifix is the primary semantic marker and MUST appear at the start of the prefix sequence (leftmost position, character 0). This fixed position ensures that the event type is always visually aligned across all log lines regardless of how many context gclifixes follow.

Curly Braces { } -- Context Tier: Describes the CONDITIONS under which the event occurred (who, where, under what privilege, in what scope, at what time). Context gclifixes are OPTIONAL. When present, they MUST follow the event gclifix. Multiple context gclifixes MAY be chained.

This design ensures that a human scanning a log file always finds the event type at a fixed column position. The context tier is supplementary metadata that enriches the line without displacing the primary signal. Emitters can start with the simpler event-only form and progressively adopt context gclifixes as their needs grow, without affecting the visual alignment of existing output. In addition, the context tier minimizes the need for extended log entry descriptors while at the same time, parsers (especially AI-powered) can build a much clearer and precise understanding of the logs without having to necessarily cross-reference additional log sources to tell the full story of who did what, when, and how.

2.3. Composition Rules

A fully composed prefixed line has the following structure:

```
line = event-prefix SP *( context-prefix SP )
       message-body

event-prefix   = "[" evt-char "]"
context-prefix = "{" ctx-char [ ":" qualifier ] "}"
qualifier      = 1*( ALPHA / DIGIT / "-" / "_" / "." )
```

The event gclifix MUST appear first. When multiple context gclifixes follow, they SHOULD be ordered from broadest scope to narrowest. The RECOMMENDED ordering is:

> Severity > Timestamp > Identity > Environment > Namespace > Execution Mode > Network

Example of a fully composed line:

```
[~] {3} {t:2025-03-14T08:12:33Z} {$:worker-7}
    {e:prod} {&} Failover to replica-3 OK
```

Example comparison demonstrating the visual alignment benefit of event-first ordering:

```
[+] {e:prod} User created successfully
[x] {#} {e:prod} Database corruption detected
[i] Module loaded
[~] {@:ssh} Connection retry succeeded
[;] {t:08:12:33Z} {$:worker-7} {&} Job #412 done
```

Note how the [ ] column remains aligned at position 0 regardless of whether context gclifixes are present or absent. This fixed-position design makes visual scanning, grep filtering, and column-based log parsing trivial.

2.4. Extensibility

This specification defines the initial gclifix registry. Additional prefix characters MAY be registered via the IANA process defined in Section 9. Any unrecognised gclifix character encountered by a consumer MUST be treated as informational and MUST NOT cause a parsing failure. In general, implementors should "be conservative in what you do, be liberal in what you accept from others" (Postel’s Law, as per [RFC9293]).

3. Event Prefix Definitions (Square Brackets)

This section defines all assigned event gclifix characters. Each entry specifies the character, its core meaning, and a brief description of its intended use.

3.1. Symbol Prefixes

3.1.1. Severity and Status

[!] Warning / Alert / Important Notice Non-fatal condition requiring attention. Execution CAN continue. Examples: security warnings, deprecation notices, potential data-loss conditions, configuration conflicts. Analogous to syslog WARNING.

[i] Informational / FYI / Neutral Detail Neutral status update. No problem indicated. Examples: configuration confirmations, module load notices, runtime context. Analogous to syslog INFO.

[+] Positive Outcome / Addition / Confirmation Successful action, creation, match, or confirmation. Examples: user created, record added, firewall rule applied, operation confirmed. Adding further clarity.

[-] Skipped / Removed / Non-Critical Omission Non-critical omission. Action bypassed or resource removed without impacting the final result. Examples: optional step skipped, temp file removed, irrelevant item filtered, not all rules matched for validation.

[~] Handled Failure / Degraded / Approximate Recoverable error: exception handling prevented a crash, but the result may be incomplete or approximate. Key distinction: unlike [x], the process did not abort; unlike [-], the failure may have affected the end result. Examples: failover succeeded, fuzzy match returned, partial import. Performed the action but low confidence and uncertainty due to the way the result was reached.

[x] Error / Fatal / Abort Unrecoverable failure. Execution halts or the specific task is abandoned. Operator must intervene. Exit code is typically non-zero. Examples: data corruption, missing critical dependency, authentication failure after retries, connection closed unexpectedly, I/O error, failed to complete operation.

3.1.2. Data Flow

[<] Import / Ingest / Read from External Source Data flowing INTO

the current process from an external source: file, API response, reading environment variable, stdin, mounted volume, network share. Complements [>].

[>] Export / Emit / Write to External Target Data flowing OUT OF the current process to an external target: file, registry, stdout, remote endpoint, mounted volume, network share. Complements [<].

[^] Upload over Network IP-based outbound (upload) transfer. Always implies a network transfer in the upload direction. Complements [v].

[v] Download over Network IP-based inbound (download) transfer. Always implies a network transfer in the download direction. Complements [^].

3.1.3. Execution Context

[#] Elevated Privileges / Root / Admin Operation requires or is using sudo, root, SYSTEM, Administrator, or equivalent escalated identity. Audit-relevant. Upwards change of privileges.

[$] User-Context Execution / Non-Privileged Running as a specific non-privileged user or service account. Distinguishes from [#]. Lateral change of privileges.

[&] Background Task / Daemon / Service Task sent to background, daemon registered, cron job, or async job dispatched. No longer blocking the foreground.

[*] Intensive Operation / Long-Running / Be Patient Heavy computation, large dataset processing, recursive time-consuming scan, compiling from source, LLM is thinking. Signals that apparent unresponsiveness is normal and expected. Pair with [%] for progress.

[%] Progress / Percentage / Processing Metric Progress indicators, throughput stats, ETA reporting. Pairs naturally with [*]. Can be emitted repeatedly on the same line (carriage return) or as successive entries.

3.1.4. Interprocess and Network

[@] Network Communication / IP Protocol Activity Any IP-protocol activity: HTTP, SSH, DNS, gRPC, MQTT, SMTP, etc. Protocol-agnostic.

[|] Pipe / Subprocess / External CLI Call Piping stdout to another

tool, spawning a child process, reading exit codes from subshells.

3.1.5. Interactive

[?] User Input Required / Prompt / Confirmation Process is blocked awaiting operator input (Y/N confirmation, passphrase entry, interactive selection). Process will not continue until input is received.

3.1.6. Lifecycle Markers

[:] Begin / Start / Open Section Start of loop iteration, beginning of log section, execution start, session open. Paired with [;]. Enables automated section extraction, tracking log entries per user or session, and duration calculation.

[;] End / Finish / Close Section End of loop iteration, close of log section, execution complete, session closed. Paired with [:]. Enables automated extraction of durations and boundaries.

[.] Clean Exit / Successful Termination / End of Process The process completed its full lifecycle and exited with code 0 (success). This gclifix SHOULD appear as the last prefixed line in a log and MAY include a final status message (e.g., uptime, records processed, resource cleanup summary). Its absence from a log that was expected to contain it is itself a diagnostic signal (crash, OOM kill, SIGKILL, power loss, etc.). Pairs with [:] (begin) and [;] (end section) to complete the lifecycle triad: colon opens, semicolon closes a section, period ends the entire story.

3.1.7. Logging

[=] Assertion / Log Access / Vaulting / Archive / Backup Asserting data to repository, accessing, or appending to log files. All data and information have been saved/filed.

3.1.8. Search and Analysis

[o] Search / Scan / Discovery / Analysis Exploratory operation: pattern search, network scan, filesystem traversal, index lookup, data analysis logging in indicator. Results not yet confirmed, in progress. Suggests a throbber, spinner, loading, wait cursor, spinning cursor.

3.1.9. Placeholder and Redaction

[_] Placeholder / Redacted / Masked / Sanitised Value exists but is

intentionally hidden, incomplete, or pending substitution. Examples: PII masked in prompts or in logs (email=g***@***.com), secret redacted (PAN 1234 12** **** 9000), template variable awaiting substitution ({{DB_HOST}}). Visually, the underscore resembles a blank/redaction line.

3.2. Alphabetic Prefixes

The following lowercase letters are assigned as event gclifixes for specific operational verbs. All alphabetic gclifixes use lowercase to avoid collision with uppercase conventions in existing logging frameworks (e.g., "[E]" for ERROR). Letters "i", "o", "v", and "x" are assigned as symbol gclifixes (Section 3) and are NOT available as alphabetic gclifixes.

[a] Append / Add to Existing Non-destructive additive write to existing resource.

[b] Backup / Snapshot / Checkpoint Creating backup or restore point. Often precedes destructive operations. Paired with [z].

[c] Cryptographic Operation Encryption, decryption, signing, verification, certificate operations, integrity hashing, encoding, decoding (Base64, URL encoding).

[d] Debug / Diagnostic / Trace Debug-level output, stack traces, variable dumps. Typically filtered in production.

[e] Environment / Configuration Context Reading/setting env vars, activating config profiles.

[f] File Operations / Filesystem Action Open, delete, move, copy, rename, release, chmod on local filesystem objects.

[g] Generate / Generative / Produce / Render Producing new artefacts: code gen, key gen, report rendering, template expansion, build output.

[h] Health Check / Heartbeat / Self-Test Liveness probes, readiness checks, self-diagnostics, keep-alive signals.

[j] Job / Scheduled Task / Queued Work Cron job, batch job, CI/CD pipeline stage, task queue.

[k] Kill / Terminate / Signal Sent SIGKILL/SIGTERM, forced shutdown, OOM kill, watchdog timeout. Distinct from [;] (graceful end).

[l] Lock / Mutex / Semaphore / Exclusive Access Acquiring/releasing

file locks, mutexes, advisory locks, DB row locks.

[m] Migration / Schema Change / Data Move DB schema migration, data migration, ETL pipeline, version-tracked structural transformation.

[n] Namespace / Scope / Context Switch K8s namespace, tenant context switch, DNS zone, variable scoping.

[p] Process / PID / Lifecycle Management Process start/stop/restart, PID reporting, fork. Distinct from [k] (termination), [&] (background).

[q] Query / Structured Lookup SQL, LDAP, GraphQL, SOAP, DNS, or any structured request-for-data. Protocol-agnostic.

[r] Refresh / Repeat / Re-pull / Request / Read-Only Refresh, repeat read, inspecting state, GET semantics. Strictly non-mutating.

[s] Save / Persist / Commit / Store Durable write: save to disk, commit transaction, persist state. Distinct from [w] and [b].

[t] Timer / Benchmark / Elapsed / Timestamp / Trigger Timing operations, benchmark results, elapsed time.

[u] Update / Upgrade / Patch In-place version change: software update, package upgrade, hotfix, firmware update.

[w] Write / Overwrite / Flush General write (may be destructive). Distinct from [a] (append) and [s] (commit).

[y] Yield / Await / Pause / Defer Intentional non-progress: yielding to scheduler, sleeping before retry, deferred execution.

[z] Undo / Revert / Rollback / Restore Returning to prior state. Often paired with [b].

3.3. Numeric Prefixes

Numeric characters serve as context-dependent markers for priority levels, selection indices, or reference footnotes.

[0] Initialisation / Reset / Zero State System down, system init, factory reset, cache clear, return to zero state.

[1] Priority 1 / Critical / Primary Reference

[2] Priority 2 / High / Secondary Reference

[3] Priority 3 / Medium / Tertiary Reference

[4] Priority 4 / Low / Reference #4

[5] Priority 5 / Informational / Reference #5

[6]-[9] Extended priority or reference indices

Context determines interpretation: in a severity context, [1] means critical (high priority) while [9] means least critical (lowest priority).

3.4. Reserved Prefixes

The following characters are explicitly RESERVED and MUST NOT be assigned semantic meaning:

[/] RESERVED -- The forward slash conflicts with Unix path separators, regex delimiters, URL path separators, and division operators. Causes parser ambiguity in JSON, shell, and regex contexts.

[\] RESERVED -- The backslash is the universal escape character in C, Python, Java, JSON, and shell. It is also the Windows path separator. Requires double-escaping ([\\]) in JSON, making it fragile and parser-hostile.

[’] RESERVED -- The single quote (apostrophe) is a string delimiter in shell, SQL, YAML, and many programming languages. Log lines containing [’] that are embedded in single-quoted contexts (shell variables, heredocs, SQL queries) would require escaping, creating avoidable fragility.

["] RESERVED -- The double quote is a string delimiter in JSON, C, Python, Java, JavaScript, shell, and CSV. A log line containing ["] embedded in a JSON field requires the quote to be escaped as [\"], which then interacts with the backslash problem. In CSV exports, ["] requires double-escaping as [""]. High risk of escaping chain reactions across multiple formats.

[,] RESERVED -- The comma is the field delimiter in CSV, which is a common log export format. While [,] is safe in JSON and shell contexts, its presence in log lines would require additional quoting logic in any CSV-based pipeline, adding unnecessary complexity. The continuation semantics it might convey are already served by [+] (adding context to the current state).

Rationale: any semantic meaning these characters could convey is already served by other assigned gclifixes. The risk of parser corruption, double-escaping bugs, and cross-platform inconsistencies far outweighs any potential benefit. Implementations encountering these prefixes in input SHOULD treat them as unrecognised informational tokens.

4. Context Prefix Definitions (Curly Braces)

Context gclifixes provide machine-parseable metadata about the conditions under which an event occurred. They are enclosed in curly braces to visually and syntactically distinguish them from event gclifixes. Context gclifixes always appear AFTER the event gclifix on a line.

4.1. Context Prefix Registry

{#} Elevated Privilege Context Active Marks the log line as produced under root/admin/SYSTEM privileges.

{$} User Identity Context Identifies the user or service account. Supports qualifier: {$:username}.

{&} Background / Async Context Line produced by a background process, daemon, or async worker -- not the foreground session.

{@} Network Context Active Operation involves a remote system or network connection. Qualifier: {@:protocol} or {@:host}.

{n} Namespace / Tenant / Scope Context Identifies which logical partition the operation belongs to. Qualifier: {n:scope_name}.

{e} Environment / Configuration Profile Context Identifies the active environment. Qualifier: {e:env_name}.

{t} Timestamp / Temporal Context Machine-parseable timestamp (ISO 8601 RECOMMENDED). Qualifier: {t:2025-03-14T08:12:33Z} or {t:+4.2s}.

{1}-{5} Severity Level Context Numeric severity for automated filtering/routing. Mirrors [1]-[5] priority scale.

{_} Redaction Context Active Marks the line as containing redacted/masked/sanitised content. Aids compliance scanning.

4.2. Qualifier Syntax

Context gclifixes MAY include an inline qualifier separated by a colon:

```
context-prefix = "{" ctx-char [ ":" qualifier ] "}"
qualifier      = 1*( ALPHA / DIGIT / "-" / "_" / "." / ":" )
```

Examples:

```
{$:svc_backup}              ; user identity
{e:production}              ; environment name
{@:ssh}                     ; network protocol
{n:us-east-1}               ; namespace/region
{t:2025-03-14T08:12:33Z}    ; ISO 8601 timestamp
{t:+4.2s}                   ; relative elapsed time
```

5. ABNF Grammar

The following ABNF grammar (as per [RFC5234]) defines the syntax of a prefixed log line:

```
prefixed-line  = event-prefix SP
                 *( context-prefix SP )
                 message

event-prefix   = "[" evt-char "]"
context-prefix = "{" ctx-char [ ":" qualifier ] "}"

evt-char       = ALPHA / DIGIT / "!" / "+" / "-" / "~" /
                 "<" / ">" / "^" / "#" / "$" / "&" / "*" /
                 "%" / "@" / "|" / "?" / ":" / ";" / "=" /
                 "." / "o" / "_"

ctx-char       = ALPHA / DIGIT / "#" / "$" / "&" / "@" /
                 "_"

qualifier      = 1*( ALPHA / DIGIT / "-" / "_" / "." / ":" )
message        = *VCHAR
SP             = %x20
```

The characters "/" (0x2F), "\" (0x5C), "'" (0x27), '"' (0x22), and "," (0x2C) are explicitly excluded from both evt-char and ctx-char.

6. Examples

6.1. Basic Event-Only Logging

The simplest form uses only event gclifixes. This is RECOMMENDED as the minimum adoption level for any CLI tool:

```
[:] === Build started ===
[i] Using toolchain gcc 13.2.0
[<] Reading source from ./src/
[o] Scanning dependencies...
[+] All dependencies resolved
[*] Compiling 43 modules...
[%] 22 of 43 modules compiled
[~] Warning in util.c:87 - implicit cast
[+] Compilation complete
[>] Wrote binary to ./build/app
[t] Total build time: 4m 12s
[;] === Build finished ===
[.] Build tool exiting. Exit code: 0
```

6.2. Composed Event + Context Logging

Production-grade logging. Note that the [ ] event gclifix is always at position 0, providing consistent visual alignment:

```
[:] {t:2025-03-14T08:00:00Z} {e:prod} {$:deploy}
    Deployment pipeline started
[u] {t:2025-03-14T08:00:03Z} {e:prod} {$:deploy} {#}
    nginx 1.25.3 -> 1.26.0
[^] {t:2025-03-14T08:00:15Z} {e:prod} {@:ssh}
    Uploading artefact to CDN
[~] {3} {t:2025-03-14T08:00:22Z} {e:prod}
    Health check timeout - retrying
[h] {t:2025-03-14T08:00:28Z} {e:prod} {$:deploy}
    Health: API ok | DB ok | Cache ok
[;] {t:2025-03-14T08:00:30Z} {e:prod} {$:deploy}
    Deployment complete (30s)
```

6.3. Interactive CLI Session

```
[i] Welcome to DataSync v3.2.0
[e] Profile loaded: production-us-east
[?] Sync direction? [1] Push [2] Pull [3] Bidi
[i] User selected: [1] Push
[b] Snapshot: sync_pre_20250314_080000
[c] Encrypting payload with AES-256-GCM...
[@] Connecting to sync.example.com:443 (TLS 1.3)
[^] Uploading 2,847 records (14.2 MB)...
[%] 67% complete (1,907 / 2,847 records)
[_] Skipped field: SSN=***-**-**** (PII masked)
[+] Sync complete: 2,847 pushed, 0 conflicts
[s] Audit log saved to /var/log/datasync/audit.log
[.] DataSync finished. Uptime: 0h 2m 14s
```

6.4. CI/CD Pipeline Output

```
[:] {t:08:00:00} {j} Pipeline #3847 (main, a1b2c3d)
[<] {t:08:00:02} {j} Checking out repository...
[q] {t:08:00:05} {j} SELECT version -> v42
[m] {t:08:00:06} {j} Migration 0043_add_user_index
[+] {t:08:00:08} {j} Migration applied (1.8s)
[*] {t:08:01:30} {j} Running test suite (847 tests)
[~] {t:08:03:45} {j} flaky_network_test: retry 1/3 OK
[+] {t:08:04:00} {j} 847/847 tests passed
[g] {t:08:04:02} {j} Building image app:a1b2c3d...
[^] {t:08:04:30} {j} Pushed to registry.example.com
[;] {t:08:04:32} {j} Pipeline #3847 done (4m 32s)
```

7. Implementation Guidance

7.1. Colour Mapping

When outputting to a terminal that supports ANSI colour codes, emitters SHOULD apply colour associations. These are RECOMMENDED defaults; implementations MAY allow user customisation:

Gclifix	Colour	Rationale
[x]	Red (ANSI 31)	Errors demand attention
[!]	Yellow (ANSI 33)	Warnings: alert, not alarm
[+]	Green (ANSI 32)	Positive: success confirm
[-]	Dim/Grey (ANSI 90)	Skipped: low-priority
[~]	Magenta (ANSI 35)	Handled failure
[i]	Cyan (ANSI 36)	Info: neutral and calm
[*]	Bold White (1;37)	Intensive: high visibility
[?]	Bold Yellow (1;33)	Prompt: must be noticed
[_]	Dim (ANSI 2)	Redacted: incomplete feel
{...}	Dim (ANSI 2)	Context: unobtrusive

Table 1

7.2. Verbosity Levels

Implementations SHOULD support filtering by gclifix. A RECOMMENDED verbosity mapping:

Level 0 (Silent): [x] only

Level 1 (Quiet): adds [!] [~]

Level 2 (Normal): adds [+] [-] [?] [:] [;]

Level 3 (Verbose): adds [i] [o] [%] [t] all alphabetic and data-flow gclifixes

Level 4 (Debug): adds [d] and all { } context gclifixes

7.3. Parsing Recommendations

Consumers SHOULD use the following regular expression (PCRE syntax) to extract gclifixes from a line:

```
\[([^\]]+)\]\s+(?:\{([^}:]+)(?::([^}]+))?\}\s+)*(.+)
```

This captures: (1) event character, (2) context character(s), (3) optional qualifier(s), and (4) message body. The event gclifix is always the first match group, reflecting its fixed position at the start of the line. Consumers MUST be prepared for lines that contain no gclifix at all (legacy output) and SHOULD pass them through unmodified.

7.4. AI Code Generation

One of the key motivations for this specification is to provide a canonical reference for large language models (LLMs) and AI code-generation tools. When an LLM generates CLI tool code, it SHOULD:

(a) Use the event gclifix vocabulary defined in Section 3 for all user-facing output messages.

(b) Select gclifixes based on semantic meaning, not aesthetic preference.

(c) Include context gclifixes (Section 4) when generating production-grade logging code.

(d) Reference this specification (by its RFC number, once assigned) in generated code comments when gclifix conventions are used.

By standardising these gclifixes, AI-generated tools will produce output that is immediately compatible with existing log aggregators, IDE extensions, and monitoring infrastructure that has adopted this specification.

8. Security Considerations

8.1. Prefix Spoofing and Trust

Gclifixes are emitter-asserted and carry no cryptographic authentication. A malicious tool could emit [+] for a failed operation or {#} without actually having elevated privileges. Consumers MUST NOT use gclifix values as the sole basis for security-critical decisions. Gclifixes are a presentation-layer convention, not a trust mechanism.

8.2. Log Injection

If user-controlled input is included in a prefixed log line without proper sanitisation, an attacker could inject crafted gclifix tokens to mislead log consumers. Emitters MUST sanitise or escape any user-supplied content appearing in the message body. The characters "[", "]", "{", and "}" in user input SHOULD be escaped or quoted to prevent false gclifix parsing.

8.3. Redaction Completeness

The [_] and {_} gclifixes signal that redaction has occurred, but they do not guarantee completeness. Downstream systems MUST NOT assume that the presence of a redaction gclifix means all sensitive data has been removed. Defence-in-depth approaches to log sanitisation remain necessary.

8.4. Parser Denial of Service

Consumers should impose reasonable limits on the number of context gclifixes per line and the length of qualifier strings to prevent resource exhaustion from maliciously crafted log lines. A RECOMMENDED maximum is 7 context gclifixes per line and 211 characters per qualifier.

8.5. Reserved Characters

The reservation of [/], [\], [’], ["], and [,] (Section 3.4) is itself a security consideration. Several of these characters are frequent vectors for path traversal, escape-sequence injection, string-delimiter confusion, and parser ambiguity attacks. Their exclusion from the gclifix registry reduces the attack surface for systems that parse prefixed log output.

9. IANA Considerations

9.1. CLI Prefix Registry

This document requests the creation of a new IANA registry called "CLI Gclifix Characters" with two sub-registries:

(a) Event Gclifix Characters (Square Brackets): Initial assignments are defined in Sections 3.1 through 3.4. New entries require Standards Action or IESG Approval.

(b) Context Gclifix Characters (Curly Braces): Initial assignments are defined in Section 4.1. New entries require Standards Action or IESG Approval.

Each registry entry MUST include: the assigned character, the bracket type (square or curly), the core meaning, a brief description, whether a qualifier is supported, and a reference to the defining document.

10. Design Rationale

This section is non-normative and explains the reasoning behind key design decisions.

10.1. Why 3 Characters?

The 3-character format (bracket + character + bracket) is the shortest token that is both visually distinctive and unambiguous to parse. Longer formats like "[INFO]" or "[WARNING]" consume valuable horizontal space and vary in width, making column-aligned log output difficult. The single-character payload maximises information density while remaining human-scannable.

10.2. Why Event-First Ordering?

An earlier revision of this draft placed context gclifixes before the event gclifix (envelope-first ordering). This was rejected in favour of event-first ordering for three reasons:

(a) Visual alignment: The event gclifix always appears at character position 0, creating a fixed visual column that humans can scan instantly regardless of context.

(b) Triage efficiency: When scanning logs, the first question is "what happened?" (event), not "under what conditions?" (context). Placing the event first matches the operator’s mental model.

(c) Progressive adoption: Most implementations will start with event-only output. Adding context gclifixes later appends to the right without shifting the event gclifix position, so existing grep patterns, column-based parsers, and muscle memory remain valid.

10.3. Why Two Bracket Types?

Overloading a single bracket pair for both events and context creates parsing ambiguity: is [#] on a given line an event ("an elevated task was performed") or context ("this line happened under elevated privileges")? The two-tier system resolves this by assigning each bracket type a distinct semantic role. Emitters can start with event-only output and adopt context gclifixes later without breaking

existing consumers.

10.4. Why Lowercase Alphabetic Gclifixes?

Many existing tools use uppercase tokens like "[E]", "[W]", "[I]" for severity levels. Using lowercase for this specification's alphabetic tier avoids collision and signals that these are part of the standardised set rather than ad-hoc tool conventions. In addition, the lowercase forms are more human-readable friendly, without being perceived as "shouting."

10.5. Why Reserve [/], [\], [’], ["], and [,]?

All five characters are overloaded across computing contexts. The forward slash serves as a path separator (Unix), URL component separator, regex delimiter, and arithmetic operator. The backslash is the universal escape character and the Windows path separator. In a JSON log line, [\] must be encoded as [\\], and even a single mishandled escaping layer corrupts the entire line. The single quote is a string delimiter in shell, SQL, and YAML; the double quote is a string delimiter in JSON, C, Python, Java, JavaScript, shell, and CSV, creating escaping chain reactions when log lines are embedded in these formats. The comma is the CSV field delimiter, and its presence in log content forces additional quoting in any CSV-based pipeline. No semantic meaning any of these characters could convey justifies the parser fragility they would introduce.

10.6. Why [_] for Redaction?

The underscore has a strong visual association with blanks and redaction lines (as in "_____"). This mnemonic makes it intuitive for both emitters choosing a gclifix and operators scanning logs. The companion {_} context gclifix enables automated compliance tools to identify log lines that have undergone PII/secret masking without parsing the message body.

10.7. Why the Name "Gclifix"?

Standards gain traction when practitioners can reference them in casual conversation. The term "gclifix" (Generic CLI Prefix) provides a single, pronounceable word for the concept, following the precedent of terms coined in other RFCs: "cookie" (RFC 6265), "nonce" (RFC 4949), "favicon" (common usage), and "JWT" (RFC 7519, pronounced "jot"). The plural "gclifixes" follows natural English morphology. Usage examples: "add gclifixes to your log output", "grep for the [o] gclifix", "we adopted both event and context gclifixes."

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

11.2. Informative References

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/info/rfc9293>.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, DOI 10.17487/RFC5424, March 2009, <https://www.rfc-editor.org/info/rfc5424>.

[RFC9116] Foudil, E. and Y. Shafranovich, "A File Format to Aid in Security Vulnerability Disclosure", RFC 9116, DOI 10.17487/RFC9116, April 2022, <https://www.rfc-editor.org/info/rfc9116>.

Appendix A. Quick-Reference Card

This appendix is non-normative. See the plaintext rendering for a formatted quick-reference card of all assigned gclifixes.

Appendix B. Revision History

draft-fragkos-cliprefix-standard-00 (March 2026): Initial submission. Defines 23 symbol event gclifixes, 22 alphabetic event gclifixes, 10 numeric event gclifixes, 5 reserved characters, and 9 context gclifix types. Introduces the term "gclifix" for the prefix vocabulary. Includes ABNF grammar, colour mapping, verbosity levels, event-first composition rules, AI code generation guidance, and security considerations.

Author’s Address

Grigorios Fragkos
Independent
Email: gfragkos@gmail.com