



CLI Text Styling and Readability Best Practices

- **Line length:** Aim for moderate line widths (roughly 80–100 characters) so output fits typical terminals without horizontal scrolling. If data is wide, consider wrapping or splitting it into multiple lines while preserving one “record” per line for scripts (e.g. a table cell can be split onto a new line) ¹. Always offer a “plain” or JSON mode (with flags like `--plain` or `--json`) for scripts ¹. Use single blank lines to separate logical sections or paragraphs; avoid “wall of text” dumps. For paged output (long lists), detect if output is a TTY and auto-pipe through a pager like `less -FIRX` ².
- **Line spacing:** Group related items closely but use blank lines to break up major sections. For example, CLI help often shows separate sections (Description, Usage, Options) each preceded by an empty line ³. Maintain consistent indentation for nested lists and sub-items to improve scanability.

Emphasis (Caps, Bold, Underline, Italics)

- **Capitalization:** Use it sparingly for structure. A common convention is **uppercase for headings or column labels** and lowercase for data rows ⁴. For example, table headers might be `ID`, `LOCATION`, etc., in UPPERCASE while entries are lower-case or capitalized naturally ⁴ ⁵. In help text or menus, section labels like `Description:`, `Usage:`, `Commands:`, `Options:` are often capitalized (first letter or all caps) and followed by indented detail ³.
- **ANSI styling:** Modern terminals support ANSI escape codes for **bold**, *italic*, and underlined text, but use them judiciously. Bold or bright text can highlight keywords or important values, and underlining can mark headings. For example, underlining a title with `=` (as in Heroku’s `regions` output) draws attention:

```
Common Runtime
=====
```

⁶. However, note that not all fonts/styles clearly show italics, and some terminals map “bold” to a brighter color. Avoid relying on italic as it is often unsupported; favor bright/bold or underline instead ⁷ ⁸.

- **Avoid gimmicks:** Do not use ALL-CAPS for entire messages or random *Markdown-style* markers (like `**stars**`) – these don’t render in plain terminals. Instead, use the terminal’s native bold (bright) or underline capabilities through ANSI codes, or simply format labels in plain text (e.g. prefix warnings with “WARNING:”).

Fonts and Cross-Terminal Appearance

- **Monospace fonts:** CLI output assumes a monospaced font for proper alignment. The tool can't control font choice – that's up to the user's terminal. Recommend users use clear, high-contrast monospaced fonts (e.g. Consolas, Menlo, Inconsolata, Fira Code). For readers with dyslexia, some find specialized fonts like **OpenDyslexic** helpful ⁹. In general, avoid using non-ASCII or complex Unicode – stick to standard glyphs so all terminals render text predictably.
- **Font size & spacing:** Since CLI tools can't set point size, encourage using sufficient terminal font size and line spacing in user settings. If you need more "room" (e.g. showing JSON), allow paging or pause to avoid crowding. Ensure any padding (spaces or tabs) lines up consistently in monospaced text.
- **Color capability:** Terminals vary (some support only 16 colors, others 256 or true-color). Prefer ANSI 4-bit colors so users can remap them via their terminal theme ¹⁰. Always provide a way to disable color (e.g. `--no-color` or respecting `NO_COLOR`) ¹¹ ¹². Do **not** assume a dark or light background – use color combinations that remain readable on both (or let the user choose their palette) ¹³.

Highlighting Important Text

- **Use color sparingly:** Color is powerful for drawing attention, but too many colors confuses. Reserve **red** for errors/fatal alerts and **yellow** or bold for warnings. A couple of consistent accent colors (e.g. green for success, cyan/magenta for key terms) is enough ⁷ ¹⁴. Don't colorize ordinary info. Ubuntu's guidelines even advise *no color by default* except for error/success ¹². Always pair color highlights with textual cues (e.g. prefix "ERROR:" or "WARN:") so colorblind users aren't lost.
- **Symbols and icons:** Simple symbols or emoji can help break monotony and draw the eye. For example, prepend `*` before a critical warning and `+` before a success message ¹⁵. In the example below, a YubiKey CLI uses `/` for emphasis:

```
$ yubikey-agent -setup
The key will be lost if the PIN and PUK are locked after 3 incorrect
tries.
...
Done! This YubiKey is secured and ready to go.
```

¹⁶. Use such symbols sparingly so they stand out (overuse reduces impact).

- **Dim or quiet text:** For less-important details (file paths, debug hints), consider a dim or gray style so the main message pops. This subtly groups information by priority without extra words. Again, offer a "verbose" mode (`-v`) to toggle these details ¹⁷.

Headings, Section Breaks, and Grouping

- **Structured sections:** Divide output into logical blocks with clear headers. CLI helps often use headings like `Commands:`, `Options:`, each followed by an indented list ³. In tables or lists, use descriptive column labels. For instance, Heroku's `regions` command outputs a table with headers ("ID", "Location", "Runtime") underlined by dashes ⁵:

\$ heroku regions		
ID	Location	Runtime
eu	Europe	Common Runtime
us	United States	Common Runtime

⁵ . Consistent spacing (e.g. two spaces between columns) ensures columns align neatly ⁴ .

- **Underlines/borders:** Use simple line separators to group related lines. A row of `---` or `==` under a heading (or between major sections) creates a visual break. Heroku's style guide shows category headers underlined with `=` ⁶ . Avoid complex ASCII art boxes, as they break with window resizing and may confuse screen readers ¹⁸ .
- **Bullet points/lists:** When listing items (e.g. subcommands or tips), use dashes or numbers. This helps scanning and works with screen readers. Indent sub-items consistently. Example CLI help uses two-space indents under each option or command description ¹⁹ .

Accessibility Considerations

- **Screen-reader friendliness:** Design output as plain, logically-ordered text. Avoid continual screen redraws (spinners) or characters that emit nontextual output – screen readers often see only text and can be disrupted by animations ²⁰ . For interactive prompts, ensure every question and option is spoken clearly (e.g. prefix prompts with words, not just symbols). Provide a *non-interactive* mode or logging option to capture output in flat text.
- **Avoid color-only cues:** Don't rely solely on color or formatting to convey meaning. WCAG advises using whitespace or text labels instead of color for headings or status ²¹ . For example, indenting or adding "ERROR:" labels alongside red color makes messages accessible to colorblind or dyslexic users. Recognize common flags like `--no-color` or the `NO_COLOR` environment variable to disable styling ¹² ¹¹ .
- **Dyslexia and vision:** Use high-contrast color choices (GitHub's CLI aligns with user-defined 4-bit palettes ¹⁰). Encourage large font or increased line spacing for readability. Recommend fonts known to aid dyslexic readers (e.g. OpenDyslexic or simple sans-serifs) ⁹ . Some terminals allow font tweaks – remind users they can customize their environment for comfort.
- **Testing:** Always test your CLI with assistive tools. Pipe output through a screen reader (e.g. Orca) to ensure it is understandable ⁹ . Check with common colorblind palettes. Strive for a high signal-to-

noise ratio: put the most important info where the eye/screen-reader will catch it first (often at the end of a message or in the error label) ²².

By following these conventions – modeled on mature CLIs like Git, Heroku, Docker, etc. – you'll create output that is **clear, consistent, and accessible**. For example, a well-formatted help screen might look like:

```
$ myapp init --help
Description:
  Create a new project.

Usage:
  myapp init <directory>

Commands:
  init    Initialize a new project in <directory>
  manage  Manage an existing project

Options:
  -h, --help      Show this help message
  -v, --version   Print version and exit
```

Figure: Clean CLI help layout using headings and indentation (adapted from CLI style guides) ³ ⁵.

Sources: Established CLI style guides and expert recommendations ⁷ ⁴ ²³ ²¹ ⁹. These emphasize human-readable, parsable output, restrained use of color/bold/underline, and accessibility (screen-reader and dyslexia considerations) in line with WCAG and tool-specific standards ¹² ¹ ¹³.

¹ ² ¹¹ ¹⁴ ¹⁵ ¹⁶ ¹⁷ ²² ²³ Command Line Interface Guidelines

<https://clig.dev/>

³ ⁴ ¹⁹ GitHub - leemunroe/cli-style-guide: A style guide for designing a consistent and usable command line interface

<https://github.com/leemunroe/cli-style-guide>

⁵ ⁶ ⁷ CLI Style Guide | Heroku Dev Center

<https://devcenter.heroku.com/articles/cli-style-guide>

⁸ Colors and formatting in the output - Better CLI

<https://bettercli.org/design/using-colors-in-cli/>

⁹ accessibility - Command line usage with dyslexia - Ask Ubuntu

<https://askubuntu.com/questions/1005182/command-line-usage-with-dyslexia>

¹⁰ ¹³ Building a more accessible GitHub CLI - The GitHub Blog

<https://github.blog/engineering/user-experience/building-a-more-accessible-github-cli/>

¹² Use of colours - CLI Guidelines - Ubuntu Community Hub

<https://discourse.ubuntu.com/t/use-of-colours/18880>

18 20 21 Best practices for inclusive CLIs - Seirdy
<https://seirdy.one/posts/2022/06/10/cli-best-practices/>