ChatGPT

Generating ANSI/Unicode Banners with Chafa

Chafa is a command-line tool that converts images (PNG, JPEG, GIF, etc.) into text-based graphics (ANSI-colored text and Unicode art) suitable for terminals [1] [2]. By default, `chafa image.png` will detect your terminal's capabilities (support for iTerm2, Kitty protocol, or Sixel) and fall back to plain ANSI symbol art if needed [3]. You can control the output format explicitly: for example, `chafa -f symbols logo.png` forces "ASCII art" output using braille and block characters [3]. Chafa supports multiple color modes (2, 8, 16, 256, or full 24-bit color); notably, the documentation **recommends 240-color mode over 256** because the lowest 16 colors vary across terminals [4]. In practice you might use:

```
chafa --colors 240 --format symbols --size ${COLUMNS}x${LINES} logo.png >
banner.txt
```

This converts `logo.png` to ANSI art sized to the current terminal width, outputting the colored text to `banner.txt`. (The `-s ${COLUMNS}x${LINES}` or `--fit-width` options ensure the art fits within the terminal [5] [6], and `--font-ratio 1/2` can adjust for your font's aspect ratio if characters appear squashed [7].)

- **Basic Chafa usage:** Run `chafa image.png` (with flags as needed) to write the ANSI/Unicode output to stdout, which you can redirect or pipe. By default it auto-detects protocols or falls back to symbols ("ANSI art") [3].
- **Output modes:** Use `-f symbols` to force plain ANSI output, or use `-f kitty`/`-f sixels` if your terminal supports those image protocols. Use `-c 240` for 240-color mode (recommended) or `-c none` for monochrome.
- **Sizing:** Use `-s ${COLUMNS}x${LINES}` or `--fit-width` to avoid line-wrapping. Chafa can also `--clear` the screen before drawing or use `--align` to center images. See the man page for more options [8] [6].

Users report that Chafa produces very detailed output (better than older ASCII tools) by using a variety of Unicode symbols (half blocks, braille, etc.) [2]. The **gallery** at the official site shows many examples of Chafa banners. Chafa also has **Python bindings** (`chafa.py`) that can generate similar output in Python apps [9]. In Python you could simply call:

```python
import subprocess
subprocess.run(["chafa", "logo.png", "-s", f"{width}x{height}"])
```

or use the `chafa.py` API:

```python
from chafa import Canvas, CanvasConfig, Loader
config = CanvasConfig(); config.width=...; config.height=...
```

```
image = Loader("logo.png"); config.calc_canvas_geometry(image.width,
image.height, font_ratio)
canvas = Canvas(config); canvas.draw_all_pixels(image.pixel_type,
image.get_pixels(), image.width, image.height, image.rowstride)
text = canvas.print().decode()
print(text)
```

This produces the ANSI/Unicode text for printing. Chafa's Python package supports Linux, macOS, and Windows [9], so it can be integrated into cross-platform CLI tools.

Embedding Chafa Output in Shell Scripts and Tools

Once you have Chafa output (ANSI text), you can embed or pipe it in shell scripts. In **Bash** you can simply call Chafa and let it print to stdout, e.g.:

```
# Directly display banner (Chafa writes to stdout):
chafa -f symbols logo.png

# Or capture it in a variable (not usually needed unless post-processing):
banner_text=$(chafa -f symbols logo.png)
echo "$banner_text"
```

For static banners, a common pattern is to pre-generate the ASCII text and embed it with a here-document, to avoid running Chafa at runtime [10]. For example:

```
cat << 'EOF'
⣿⣿⣿⣿⣿⣿⡏  ⢹⣿⡇
⣿⣿⣿⣿⣿⣿⣿  ⢹⣿⡏
... rest of pre-generated ANSI art ...
EOF
```

Using `cat << 'EOF'` (with quotes) preserves all characters literally [10]. Alternatively, you can store the output of `chafa` in a text file (e.g. `chafa logo.png > banner.txt`) and later in your script do `cat banner.txt`. This ensures the banner is printed exactly and can include ANSI color codes.

In **PowerShell**, the idea is similar. You can save the Chafa output to a `.txt` file and then read it back. For example:

```
# Read a pre-generated banner and print it
$banner = Get-Content -Raw -Path ".\banner.txt"
Write-Host $banner
```

Or embed as a here-string:

```
$banner = @"
⣿⣿⣿⣿⣿⣿⡏  ⣹⣿⣿⡇
⣿⣿⣿⣿⣿⣿   ⣹⣿⣿⠁
... (ANSI text) ...
"@
Write-Host $banner
```

PowerShell's here-strings keep line breaks and symbols intact [11]. (Note: if you capture Chafa output via `Chafa.exe | Out-File` on Windows, be aware some versions had issues freezing the shell [12].)

In **Python** scripts or CLI programs, if you don't use `chafa.py`, you can spawn `chafa` similarly. For example:

```python
import subprocess
result = subprocess.run(["chafa", "logo.png", "-f", "symbols"],
capture_output=True)
print(result.stdout.decode())
```

Or install and use the `chafa.py` library directly, as shown above. This avoids shell-escaping issues and works on any OS supported by the library [9].

Pre-generated vs. Dynamic Conversion

There are trade-offs to consider. **Pre-generating** the banner (e.g. running Chafa ahead-of-time and storing the text) makes your CLI self-contained (no Chafa dependency at runtime) and faster to start. You can embed the static ASCII art directly in your script or as an asset file. This is fine if the banner is fixed and you're not concerned about adapting to terminal size or color depth. For example, many tools just `cat` a fixed ANSI-logo at startup.

By contrast, **dynamic conversion** (running Chafa each time) lets you tailor the output to the current environment (terminal width, color support, etc.). You can, for instance, detect `$COLUMNS` and `$LINES` in Bash or use Python's `shutil.get_terminal_size()`, and then run `chafa -s ${COLUMNS}x${LINES}` to scale the image. Chafa also has a `--probe` mode to auto-detect the terminal's capabilities. Dynamic is more flexible but requires Chafa to be installed (and adds startup overhead). A compromise is to generate a few variants (e.g. different widths or color modes) and choose one at runtime.

As a best practice, if you pre-generate, keep the raw ANSI text in UTF-8 and include it via a literal or `cat`. If dynamic, ensure you call Chafa before any other output or prompts, and consider `stty` or environment variables so the terminal size is correctly determined. Always test in a plain SSH session and in a multiplexer (tmux/screen) if you need to support those.

ANSI/Unicode Compatibility (Windows, macOS, Linux)

ANSI color and Unicode support varies by OS and terminal emulator:

- **Linux/macOS terminals:** Most modern emulators (GNOME Terminal, iTerm2, Konsole, etc.) support 24-bit ANSI (TrueColor) and full Unicode (including braille patterns, half blocks, etc.). Chafa's TrueColor output will display correctly on these by default [1]. Ensure your `$TERM` is set properly (e.g. `xterm-256color`) so programs detect 24-bit or 256-color support.
- **Windows:** Older Windows consoles (pre-2016) had very limited ANSI support. However, **Windows Terminal** (and modern versions of cmd/PowerShell) now support 24-bit color and even the Sixel graphics protocol. In fact, Windows Terminal 1.22+ added sixel image support, and chafa can output sixels to leverage that [13]. (For example, `chafa -f sixels picture.png` will print a sixel image in Windows Terminal.) If you only use ANSI symbols, note that Windows 10+ Console has reasonably good ANSI support, but older Windows 7/8 consoles will show weird chars. It's best to test on your lowest-common-denominator. The [Windows Terminal release notes] even mention using chafa for sixel encoding [13].
- **Multiplexers:** If running under tmux/screen, make sure they're configured to allow passthrough of sixels or Kitty images. (For tmux, `set -g allow-passthrough on` and `update-environment` are needed, see Yazi docs [14] [15].)

Chafa can output in various modes to match terminal capabilities [1], and you should choose a color mode supported by your target. For example, use `-c 240` (240 colors) which is widely supported and recommended [4]. You can also detect `$COLORTERM` or terminfo and fallback to `-c 16` if full color isn't available.

Ensuring Proper Spacing and Layout

Text banners rely on *monospaced fonts* and proper width. Some tips:

- **Font Aspect Ratio:** Chafa assumes a default font aspect of 1:2 (width:height). If your terminal font is unusually wide/narrow, use `--font-ratio` to adjust (e.g. `--font-ratio 12/24`) [7]. This ensures the ASCII art isn't squished or stretched.
- **Monospace Fonts:** Verify your terminal uses a monospaced font. If it uses variable-width fonts or has custom glyph widths (as some Windows fonts can), the layout will break. Most CLI environments default to monospace.
- **Line Length:** Make sure the banner's width doesn't exceed the terminal. Use `chafa -s $(tput cols)x$(tput lines)` or set `--margin-right 1` to keep a safety buffer [6]. If the output line is wider than the screen, it will wrap, mangling the image. One trick: after generating, pipe through `fold -s -w $(tput cols)` to enforce wrapping.
- **Terminal Resizing:** If the user resizes the window after printing the banner, the art won't reflow. There's no perfect fix; just design the banner to fit typical sizes or center it with `--align center`. Alternatively, clear the screen (`--clear`) before printing so the user cannot scroll up and see a broken wrap.
- **Line Wrapping:** You generally *cannot* disable wrap in a portable way. Rely on sizing instead. If you *really* need to prevent any wrapping, you could use an escape (like CSI ?7l to disable wrap on some

terminals), but that's not standard and can be confusing. Better to just make the output <= terminal width.

- **Combining Characters/Emoji:** Avoid fancy emojis or full-width CJK chars in the banner, as their behavior varies (some terminals count them as 2 columns, some as 1). Note that Windows Terminal recently improved grapheme cluster support [16], but older terminals might misalign. Stick to simple Unicode blocks and braille symbols (U+2800–U+28FF, U+2580–U+259F) which are universally single-cell in monospaced fonts.
- **Color Legibility:** Remember that terminal background color affects visibility. Use Chafa's `--bg COLOR` to pick a background (e.g. black) if needed [17]. Test both light and dark terminal themes.

Avoiding Issues with Wrapping and Resize

To prevent common issues: always generate banners to *fit* the environment. For example, in Bash you might do:

```
width=$(tput cols)
height=$(tput lines)
chafa -f symbols --size ${width}x${height} banner.png
```

This ensures the output is no wider/taller than the terminal. In Python, use `os.get_terminal_size()` similarly. Also use `--clear` at the start so old content is removed: `clear; chafa ...` in scripts. If your tool prints a prompt or menu after the banner, add a blank line afterward to separate and avoid accidental overwrite.

If wrapping still occurs, users can sometimes scroll horizontally (`less -S` style) but that's not ideal for a splash banner. The safest approach is simply designing your image for a maximum width (e.g. 80 or 100 columns) and testing in the narrowest expected terminal.

Example Tools Using Chafa/ASCII Banners

A few terminal apps have adopted image-as-ASCII strategies:

- **Yazi (Rust TUI file manager):** Yazi's documentation notes that it auto-selects the best image preview method, *falling back to ASCII via Chafa* if no graphics protocol is available [18] [19]. In other words, Yazi will use Chafa as a last-resort to display pictures as ANSI art (and explicitly marks Chafa as "the last fallback resort" [19]). This shows a real-world use: Yazi tries native image protocols (Kitty, iTerm, sixel, Überzug) and only if none work does it revert to a Chafa-rendered banner.
- **Basht (Bash TUI file manager):** The Basht project can optionally use Chafa for image previews (alongside other methods like Kitty or Uberzug) [20]. While Basht's logos/examples aren't purely ASCII, it demonstrates that scripts integrate Chafa in pipelines for showing images in-terminal.
- **Neofetch-style banners:** Many system info tools (like neofetch) display an ASCII art logo at startup. Usually these are static text (Figlet fonts) rather than images. However, you could enhance a tool by generating a PNG of your logo and running it through Chafa for a richer look. The approach is the same as above.

- **Terminal demos:** The [Windows Terminal blog] itself mentions using `chafa` to convert PNGs to sixels for display [13], implying you can pipe Chafa into the terminal to show images. Tools like `img2sixel` or `chafa` are now standard ways to show pictures in modern consoles.

In practice, any CLI program that prints a startup splash (a logo or title) could incorporate a Chafa banner. For example:

```
main() {
    # Clear and show image banner
    tput clear
    chafa --symbols logo.png
    echo "Welcome to MyTool v1.0"
    ... rest of script ...
}
```

Be mindful that while such banners look cool, they add complexity and may not work on every setup (remote SSH to a minimal console, etc.). Always provide a way to skip or disable it if needed (e.g. a `--no-banner` flag).

In summary: To embed Chafa-generated art as a CLI banner, you either call Chafa at runtime (ensuring the terminal is large enough and supports ANSI colors [1] [4]) or pre-generate the text and include it as static content (using a here-doc in Bash [10] or here-string in PowerShell [11]). Adjust color mode and font aspect flags for consistency, and test across different environments. With these strategies, your CLI tool can greet users with rich, image-like banners entirely in text.

Sources: Official Chafa documentation and examples [3] [7] [4] [1]; power/shell scripting Q&A [11] [10]; Windows Terminal blog (sixel support) [13]; and user reports (Yazi docs, blog posts) [18] [2]. These illustrate how Chafa converts images and how its output can be integrated into CLI apps.

---

[1] 10 Cool Command Line Tools For Your Linux Terminal
https://www.ucartz.com/clients/knowledgebase/1824/10-Cool-Command-Line-Tools-For-Your-Linux-Terminal.html?language=hebrew

[2] Cutting videos in the terminal with chafa and ffmpeg
https://wonger.dev/posts/chafa-ffmpeg-progress

[3] [4] [5] [6] [7] [8] [17] Man page for Chafa
https://hpjansson.org/chafa/man/

[9] chafa.py · PyPI
https://pypi.org/project/chafa.py/

[10] bash - How do you output a multi-line string that includes slashes and other special characters? - Unix & Linux Stack Exchange
https://unix.stackexchange.com/questions/678930/how-do-you-output-a-multi-line-string-that-includes-slashes-and-other-special-ch

11 powershell - How do I output ASCII Art to console? - Stack Overflow
https://stackoverflow.com/questions/35022078/how-do-i-output-ascii-art-to-console

12 Piping and redirection are broken in Windows · Issue #283 · hpjansson/chafa · GitHub
https://github.com/hpjansson/chafa/issues/283

13 16 Windows Terminal Preview 1.22 Release - Windows Command Line
https://devblogs.microsoft.com/commandline/windows-terminal-preview-1-22-release/

14 15 18 19 Image Preview | Yazi
https://yazi-rs.github.io/docs/image-preview/

20 Basht, a tui file manager bash script that supports image previews (among other features) : r/linux
https://www.reddit.com/r/linux/comments/1baqc42/basht_a_tui_file_manager_bash_script_that/