Gemini Workflow User's Guide

Version: 1.0.2-Beta Date: 2025-10-09

--- Revision History ---

[1.0.2-Beta] - 2025-10-09

Added

- Added "Flexible and Data-Driven" section to explain new architectural

patterns like CLI overrides and JSON inheritance.

[1.0.1-Beta] - 2025-09-29

Added

- Added "Keeping Logic Organized" to the Core Philosophy to explain

the new architectural principle.

- Added "Brainstorming and Exploration" to the task lifecycle to

describe the new optional exploratory protocol.

[1.0.0-Beta] - 2025-09-28

Added

- Added "The Most Important Principle: Safety First" to explain the new

"Process Over Preference" principle and its user benefits.

Changed

- Rewrote "The Lifecycle of a Task" to reflect the new, mandatory

user-gated step between AI analysis and implementation plan creation.

- Updated the "Definitive State Initialization" section to include the

This guide provides a human-readable explanation of the development workflow used by the Gemini AI agent for the Contest Log Analyzer project. For the complete, machine-focused technical specification, please see `AIAgentWorkflow.md`.

Core Philosophy: A Safe and Predictable Partnership

Our collaboration is built on a few key ideas to ensure that the development process is safe, reliable, and that you are always in control.

- **The Most Important Principle: Safety First:** Our workflow prioritizes safety, reliability, and user control over speed. While a strict, step-by-step process might seem slower, it is a critical feature designed to protect the project from errors, prevent file corruption, and save time by avoiding long, confusing debugging sessions. The AI will always follow the established protocols without taking shortcuts.
- **You Are in Control:** The AI will not take unrequested actions. The workflow is designed as a state machine where you, the user, provide the commands that move us from one stage to the next. The AI will perform an action and then halt, waiting for your next instruction.
- **Keeping Logic Organized:** We keep the user interface code separate from the core analysis logic. This makes the system more robust and easier to maintain or even upgrade with a new interface (like a web app) in the future.
- **Flexible and Data-Driven:** The system is designed to be adaptable. Rule sets for similar contests can be reused to speed up development. Furthermore, command-line flags give you the power to analyze the same log file in different ways (e.g., scoring an IARU log as a standard entry or a WRTC entry) without ever needing to modify the original log file.
- **Surgical Changes:** The AI is forbidden from rewriting files from scratch. All changes are small, surgical modifications to the last known-good version of a file. This prevents accidental regressions and makes every change easy to review.

The Lifecycle of a Task

Every development task, from a simple bug fix to a new feature, follows a strict, predictable pattern. 0. **Brainstorming and Exploration (Optional):** For new or complex ideas where the requirements aren't fully defined, we can start with an "Exploratory Mode." This is a less formal discussion to scope out the task before we begin the formal lifecycle below.

1. **Request and Analysis:** You start by providing a task, like fixing a bug or adding a feature. The AI will analyze the request, ask clarifying questions if needed, and provide its initial findings.
2. **The User Gate:** After the analysis is complete, the AI will **stop** and **ask for your permission** to create an implementation plan. The AI will never create a plan without your explicit command. This is a critical

checkpoint that ensures your intent is fully understood before the detailed planning work begins.

3. **Implementation Plan:** Once you grant permission, the AI will produce a formal **Implementation Plan**. This is a detailed, multi-part document that describes exactly which files will be changed and shows the precise line-by-line modifications (`diffs`) for your review. Nothing happens without a plan.
4. **Approval:** You review the plan. If it's correct, you approve it with the single keyword `Approved`. This is the trigger for the AI to start generating files.
5. **Execution and Delivery:** The AI will then generate each file, one by one. For each file, it will ask you to `Confirm` before generating it and `Acknowledge` after it has been delivered. This per-file checkpoint ensures maximum context integrity.

Handling Errors and Resets

When things go wrong, we rely on a ”hard reset” to get back to a known-good state. A **Definitive State Initialization** is our ”hard reset” button. It's used at the start of a project or any time the AI's context seems corrupted.

1. You request the initialization.
2. You upload ”bundle” files containing all the project's code, documentation, and data.
3. After a successful integrity check, the AI will ask for a final `Confirmed` prompt before it purges its memory and re-learns the project from only your files. This is a critical safety step.
4. The AI confirms the new state by listing all the files it has loaded.