Appendix E - AI Agents on the CLI

Introduction

The developer's command line, long a bastion of precise, imperative commands, is undergoing a profound transformation. It is evolving from a simple shell into an intelligent, collaborative workspace powered by a new class of tools: AI Agent Command-Line Interfaces (CLIs). These agents move beyond merely executing commands; they understand natural language, maintain context about your entire codebase, and can perform complex, multi-step tasks that automate significant parts of the development lifecycle.

This guide provides an in-depth look at four leading players in this burgeoning field, exploring their unique strengths, ideal use cases, and distinct philosophies to help you determine which tool best fits your workflow. It is important to note that many of the example use cases provided for a specific tool can often be accomplished by the other agents as well. The key differentiator between these tools frequently lies in the quality, efficiency, and nuance of the results they are able to achieve for a given task. There are specific benchmarks designed to measure these capabilities, which will be discussed in the following sections.

Claude CLI (Claude Code)

Anthropic's Claude CLI is engineered as a high-level coding agent with a deep, holistic understanding of a project's architecture. Its core strength is its "agentic" nature, allowing it to create a mental model of your repository for complex, multi-step tasks. The interaction is highly conversational, resembling a pair programming session where it explains its plans before executing. This makes it ideal for professional developers working on large-scale projects involving significant refactoring or implementing features with broad architectural impacts.

Example Use Cases:

1. **Large-Scale Refactoring:** You can instruct it: "Our current user authentication relies on session cookies. Refactor the entire codebase to use stateless JWTs, updating the login/logout endpoints, middleware, and frontend token handling." Claude will then read all relevant files and perform the coordinated changes.

2. **API Integration:** After being provided with an OpenAPI specification for a new weather service, you could say: "Integrate this new weather API. Create a service module to handle the API calls, add a new component to display the weather, and update the main dashboard to include it."
3. **Documentation Generation**: Pointing it to a complex module with poorly documented code, you can ask: "Analyze the ./src/utils/data_processing.js file. Generate comprehensive TSDoc comments for every function, explaining its purpose, parameters, and return value."

Claude CLI functions as a specialized coding assistant, with inherent tools for core development tasks, including file ingestion, code structure analysis, and edit generation. Its deep integration with Git facilitates direct branch and commit management. The agent's extensibility is mediated by the Multi-tool Control Protocol (MCP), enabling users to define and integrate custom tools. This allows for interactions with private APIs, database queries, and execution of project-specific scripts. This architecture positions the developer as the arbiter of the agent's functional scope, effectively characterizing Claude as a reasoning engine augmented by user-defined tooling.

Gemini CLI

Google's Gemini CLI is a versatile, open-source AI agent designed for power and accessibility. It stands out with the advanced Gemini 2.5 Pro model, a massive context window, and multimodal capabilities (processing images and text). Its open-source nature, generous free tier, and "Reason and Act" loop make it a transparent, controllable, and excellent all-rounder for a broad audience, from hobbyists to enterprise developers, especially those within the Google Cloud ecosystem.

Example Use Cases:

1. **Multimodal Development:** You provide a screenshot of a web component from a design file (gemini describe component.png) and instruct it: "Write the HTML and CSS code to build a React component that looks exactly like this. Make sure it's responsive."
2. **Cloud Resource Management:** Using its built-in Google Cloud integration, you can command: "Find all GKE clusters in the production project that are running versions older than 1.28 and generate a gcloud command to upgrade them one by one."
3. **Enterprise Tool Integration (via MCP):** A developer provides Gemini with a custom tool called get-employee-details that connects to the company's internal HR API. The prompt is: "Draft a welcome document for our new hire. First, use

the get-employee-details --id=E90210 tool to fetch their name and team, and then populate the welcome_template.md with that information."

4. **Large-Scale Refactoring**: A developer needs to refactor a large Java codebase to replace a deprecated logging library with a new, structured logging framework. They can use Gemini with a prompt like: Read all *.java files in the 'src/main/java' directory. For each file, replace all instances of the 'org.apache.log4j' import and its 'Logger' class with 'org.slf4j.Logger' and 'LoggerFactory'. Rewrite the logger instantiation and all .info(), .debug(), and .error() calls to use the new structured format with key-value pairs.

Gemini CLI is equipped with a suite of built-in tools that allow it to interact with its environment. These include tools for file system operations (like reading and writing), a shell tool for running commands, and tools for accessing the internet via web fetching and searching. For broader context, it uses specialized tools to read multiple files at once and a memory tool to save information for later sessions. This functionality is built on a secure foundation: sandboxing isolates the model's actions to prevent risk, while MCP servers act as a bridge, enabling Gemini to safely connect to your local environment or other APIs.

Aider

Aider is an open-source AI coding assistant that acts as a true pair programmer by working directly on your files and committing changes to Git. Its defining feature is its directness; it applies edits, runs tests to validate them, and automatically commits every successful change. Being model-agnostic, it gives users complete control over cost and capabilities. Its git-centric workflow makes it perfect for developers who value efficiency, control, and a transparent, auditable trail of all code modifications.

Example Use Cases:

1. **Test-Driven Development (TDD):** A developer can say: "Create a failing test for a function that calculates the factorial of a number." After Aider writes the test and it fails, the next prompt is: "Now, write the code to make the test pass." Aider implements the function and runs the test again to confirm.
2. **Precise Bug Squashing:** Given a bug report, you can instruct Aider: "The calculate_total function in billing.py fails on leap years. Add the file to the context, fix the bug, and verify your fix against the existing test suite."
3. **Dependency Updates:** You could instruct it: "Our project uses an outdated version of the 'requests' library. Please go through all Python files, update the import statements and any deprecated function calls to be compatible with the latest version, and then update requirements.txt."

GitHub Copilot CLI

GitHub Copilot CLI extends the popular AI pair programmer into the terminal, with its primary advantage being its native, deep integration with the GitHub ecosystem. It understands the context of a project *within GitHub*. Its agent capabilities allow it to be assigned a GitHub issue, work on a fix, and submit a pull request for human review.

Example Use Cases:

1. **Automated Issue Resolution:** A manager assigns a bug ticket (e.g., "Issue #123: Fix off-by-one error in pagination") to the Copilot agent. The agent then checks out a new branch, writes the code, and submits a pull request referencing the issue, all without manual developer intervention.
2. **Repository-Aware Q&A:** A new developer on the team can ask: "Where in this repository is the database connection logic defined, and what environment variables does it require?" Copilot CLI uses its awareness of the entire repo to provide a precise answer with file paths.
3. **Shell Command Helper:** When unsure about a complex shell command, a user can ask: gh? find all files larger than 50MB, compress them, and place them in an archive folder. Copilot will generate the exact shell command needed to perform the task.

Terminal-Bench: A Benchmark for AI Agents in Command-Line Interfaces

Terminal-Bench is a novel evaluation framework designed to assess the proficiency of AI agents in executing complex tasks within a command-line interface. The terminal is identified as an optimal environment for AI agent operation due to its text-based, sandboxed nature. The initial release, Terminal-Bench-Core-v0, comprises 80 manually curated tasks spanning domains such as scientific workflows and data analysis. To ensure equitable comparisons, Terminus, a minimalistic agent, was developed to serve as a standardized testbed for various language models. The framework is designed for extensibility, allowing for the integration of diverse agents through containerization or direct connections. Future developments include enabling massively parallel evaluations and incorporating established benchmarks. The project encourages open-source contributions for task expansion and collaborative framework enhancement.

Conclusion

The emergence of these powerful AI command-line agents marks a fundamental shift in software development, transforming the terminal into a dynamic and collaborative environment. As we've seen, there is no single "best" tool; instead, a vibrant ecosystem is forming where each agent offers a specialized strength. The ideal choice depends entirely on the developer's needs: Claude for complex architectural tasks, Gemini for versatile and multimodal problem-solving, Aider for git-centric and direct code editing, and GitHub Copilot for seamless integration into the GitHub workflow. As these tools continue to evolve, proficiency in leveraging them will become an essential skill, fundamentally changing how developers build, debug, and manage software.

References

1. Anthropic. *Claude*. https://docs.anthropic.com/en/docs/claude-code/cli-reference
2. Google Gemini Cli https://github.com/google-gemini/gemini-cli
3. Aider. https://aider.chat/
4. GitHub *Copilot CLI* https://docs.github.com/en/copilot/github-copilot-enterprise/copilot-cli
5. Terminal Bench: https://www.tbench.ai/