DocCompiler.ai

USER GUIDE

Gemini Cli User Guide

Installation, Usage & Quick Reference

A Cyber Panda Solutions LLC product

Version 3.1 2026-05-07 @ a809bc7 Generated by DocCompiler.ai

Always-updated version at doccompiler.ai/google-gemini/gemini-cli

Source code analysis by Claude Opus 4.6 | 1M token context

About This Document

What Is This?

This is a comprehensive, source-code-verified **User Guide** for **Gemini Cli**, generated by DocCompiler.ai using Claude Opus 4.6 with 1M-token context. Every command, configuration option, and behavior documented here was verified against the actual source code.

The Definitive Reference

This document is designed to be the single source of truth for Gemini Cli. Rather than piecing together scattered README fragments, outdated wiki pages, forum posts, and version-specific screenshots, you can rely on this as a verified, structured reference backed by automated code review.

Optimized for AI Assistants

This document is structured for consumption by AI assistants. Feed this PDF into your preferred LLM for accurate, grounded answers about Gemini Cli — instead of having the model guess or hallucinate from incomplete context.

Version & Updates

Generated on **2026-05-07** (version **3.1**). This document reflects the source code at commit `a809bc7`. The always-updated version is permanently available at **doccompiler.ai/google-gemini/gemini-cli**. Do not print or download for offline use — this document is updated when the source code changes.

Our Mission

DocCompiler.ai aims to be the definitive wiki for the world's most important open-source projects. Every document is overkill by design: exhaustive, fact-checked, and built to replace the scattered tribal knowledge that slows teams down.

Table of Contents

Gemini CLI User Guide

Gemini CLI is Google's AI-powered command-line tool that brings the full capabilities of the Gemini family of models directly into your terminal. It reads, writes, and executes code; manages files; runs shell commands; searches the web; and connects to external services through MCP servers and extensions. Whether you need to scaffold a new project, debug a tricky issue, refactor a module, or automate a multi-step workflow, Gemini CLI acts as an AI coding agent that works alongside you in any directory.

This guide covers everything you need to install, configure, and use Gemini CLI for everyday development tasks.

Quick Start

Get up and running in under five minutes.

1. Install

bash

```
npm install -g @anthropic-ai/gemini-cli
```

> **Note**
>
> Gemini CLI requires **Node.js 20 or later**. Verify with `node --version`.

2. Authenticate

On first run, Gemini CLI walks you through authentication. The fastest path:

bash

```
gemini
```

Select **Sign in with Google** when prompted. A browser window opens for OAuth consent. Once approved, the CLI stores credentials securely in your OS keychain.

3. Ask Your First Question

bash

```
gemini -p "What files are in this directory and what do they do?"
```

Or launch interactively and type your request:

bash

```
gemini
&gt; Explain how the authentication middleware works in this project
```

4. Let It Work

Gemini reads your codebase, runs shell commands, edits files, and reports back. You approve each action (or enable auto-approval modes for faster iteration).

Installation

System Requirements

Requirement	Minimum
Node.js	20.0.0+
npm	10+ (ships with Node 20)
OS	macOS, Linux, Windows
Network	Internet access for Gemini API

Install via npm

bash

```
npm install -g @google/gemini-cli
```

Verify installation:

bash

```
gemini --version
```

Update to Latest

bash

```
npm update -g @google/gemini-cli
```

> **Tip**
> Gemini CLI checks for updates automatically on launch. Control this behavior with the `general.enableAutoUpdate` setting.

Uninstall

bash

```
npm uninstall -g @google/gemini-cli
```

Authentication

Gemini CLI supports four authentication methods. Choose based on your environment.

Sign in with Google (Default)

Best for individual developers. Opens a browser for Google OAuth:

bash

```
gemini
# Select "Sign in with Google" at the auth prompt
```

Credentials are stored securely in your OS keychain via `gemini-cli-api-key` service.

Gemini API Key

Best for CI/CD, scripts, and environments without a browser:

bash

```
export GEMINI_API_KEY="your-api-key-here"
gemini -p "Run the test suite"
```

Get your API key from Google AI Studio.

Vertex AI

For Google Cloud users who need enterprise controls:

bash

```
export GOOGLE_CLOUD_PROJECT="your-project-id"
export GOOGLE_CLOUD_LOCATION="us-central1"
gemini
```

Or use express mode with just an API key:

bash

```
export GOOGLE_API_KEY="your-google-api-key"
gemini
```

Application Default Credentials (ADC)

For compute environments (Cloud Shell, GCE, Cloud Run):

bash

```
gcloud auth application-default login
gemini
```

> **Important**
>
> When using **Gemini API** auth, the `GEMINI_API_KEY` environment variable must be set. When using **Vertex AI**, you must set either `GOOGLE_CLOUD_PROJECT` + `GOOGLE_CLOUD_LOCATION` or `GOOGLE_API_KEY`. The CLI validates these on startup and gives clear error messages if they're missing.

Switching Auth Methods

Use the `/auth` slash command during a session:

```
/auth
```

This presents the authentication selector to switch methods without restarting.

Configuration

Settings Files

Gemini CLI uses a layered settings system. Settings merge from broadest to most specific scope:

Scope	Location	Purpose
System	`GEMINI_CLI_SYSTEM_SETTINGS_PATH` env var	Organization-wide defaults
System Defaults	`GEMINI_CLI_SYSTEM_DEFAULTS_PATH` env var	Fallback defaults
User	`~/.gemini/settings.json`	Personal preferences
Workspace	`.gemini/settings.json` (in project root)	Project-specific settings

More-specific scopes override broader ones. All settings files use the same JSON schema.

> **Tip**
>
> Add `"$schema": "https://raw.githubusercontent.com/google-gemini/gemini-cli/main/schemas/settings.schema.json"` to the top of your `settings.json` for IDE autocompletion and validation.

Key Settings Categories

General:

```json
{
  "general": {
    "vimMode": false,
    "preferredEditor": "code",
    "defaultApprovalMode": "default",
    "checkpointing": { "enabled": false },
    "maxAttempts": 10,
    "enableAutoUpdate": true,
    "enableNotifications": false,
    "sessionRetention": {
      "enabled": true,
      "maxAge": "30d"
    }
  }
}
```

Model:

```json
{
  "model": {
    "name": "gemini-2.5-pro",
    "compressionThreshold": 0.5,
```

```
    "maxSessionTurns": -1,
    "disableLoopDetection": false
  }
}
```

UI:

```json
{
  "ui": {
    "theme": "Dark",
    "hideBanner": false,
    "showLineNumbers": true,
    "compactToolOutput": true,
    "loadingPhrases": "off",
    "errorVerbosity": "low",
    "inlineThinkingMode": "off",
    "useAlternateBuffer": false
  }
}
```

Privacy:

```json
{
  "privacy": {
    "usageStatisticsEnabled": true
  }
}
```

GEMINI.md -- Project Context

Create a `GEMINI.md` file in your project root to give Gemini persistent context about your project:

```bash
gemini -p "/init"
```

Or use the `/init` slash command in a session. This analyzes your project and generates a starter `GEMINI.md`.

GEMINI.md files follow a hierarchy:

Location	Scope
`~/.gemini/GEMINI.md`	Global (all projects)
`./GEMINI.md`	Project root
`./src/GEMINI.md`	Subdirectory-specific

All applicable GEMINI.md files are loaded into the model's context at the start of each session.

Environment Variables

Key environment variables for customizing behavior:

Variable	Purpose
`GEMINI_API_KEY`	API key for Gemini API auth
`GOOGLE_CLOUD_PROJECT`	GCP project for Vertex AI
`GOOGLE_CLOUD_LOCATION`	GCP region for Vertex AI
`GOOGLE_API_KEY`	Google API key (Vertex express mode)
`GEMINI_MODEL`	Override the default model
`GEMINI_SANDBOX`	Sandbox mode: `docker`, `podman`, or empty
`GEMINI_SANDBOX_IMAGE`	Custom Docker image for sandbox
`GEMINI_CONFIG_DIR`	Override config directory path
`GEMINI_DEBUG_LOG_FILE`	Path for debug log output
`DEBUG`	Enable debug logging
`HTTPS_PROXY` / `HTTP_PROXY`	Proxy configuration
`NO_PROXY`	Proxy bypass list

Core Usage

Interactive Mode

Launch without arguments to enter the interactive TUI:

bash

```
gemini
```

Type your request and press Enter. The model reads your project, thinks through the problem, and proposes actions. You approve or reject each tool call.

Keyboard shortcuts:

Key	Action
`Enter`	Send message
`Escape`	Cancel current operation
`?`	Show shortcuts
`Tab`	Toggle Plan/Act mode
`Ctrl+C`	Interrupt / Exit

Non-Interactive Mode

Execute a single prompt and exit:

bash

```
gemini -p "Add error handling to the login function"
```

Use `--prompt-interactive` (`-pi`) to run a prompt then stay in interactive mode:

bash

```
gemini -pi "Set up the project, then let me review"
```

Piped Input

Pipe content directly:

bash

```
cat error.log | gemini -p "What went wrong here?"
git diff | gemini -p "Review this diff"
```

Output Formats

bash

```
# Default text output
gemini -p "List all TODO comments"

# JSON output for scripting
gemini --output-format json -p "List all TODO comments"

# Raw output (no UI chrome)
gemini --raw-output -p "Generate a Dockerfile"
```

> **Warning**
>
> Raw output mode (`--raw-output`) sends model output directly to stdout without safety filtering. You must acknowledge the risk with `--accept-raw-output-risk`.

Approval Modes

Control how much autonomy Gemini CLI has when executing actions.

Default Mode

Every tool call requires your explicit approval:

bash

```
gemini
```

You see each proposed action and choose: approve, reject, or edit.

Auto-Edit Mode

Automatically approves file edits but still asks for shell commands:

bash

```
gemini --approval-mode auto_edit
```

YOLO Mode

Auto-approves everything. Use for trusted, well-scoped tasks:

bash

```
gemini --yolo
# or
gemini --approval-mode yolo
```

> **Caution**
>
> YOLO mode executes all tool calls without confirmation, including shell commands. Only use this in sandboxed environments or when you fully trust the task scope. YOLO mode can only be enabled via the command line, not via settings files.

Plan Mode

Read-only mode. Gemini can analyze and plan but cannot make changes:

bash

```
gemini --approval-mode plan
```

Toggle between Plan and Act modes during a session with `Tab` or the `/plan` command.

Plan mode routing: When using an `auto` model, Gemini CLI automatically routes to the Pro model during planning and the Flash model during implementation for cost efficiency.

Setting a Default

json

```
{
  "general": {
    "defaultApprovalMode": "auto_edit"
  }
}
```

Valid values: `"default"`, `"auto_edit"`, `"plan"`. YOLO is intentionally excluded from settings.

Models

Available Models

Model	Tier	Best For
`gemini-3-pro-preview`	Pro	Complex reasoning, architecture, debugging

`gemini-3-flash-preview`	Flash	Fast responses, simple tasks
`gemini-2.5-pro`	Pro	Stable, production-ready
`gemini-2.5-flash`	Flash	Fast, cost-effective
`gemini-2.5-flash-lite`	Flash Lite	Lightweight internal tasks
`auto-gemini-3`	Auto	Routes between Gemini 3 Pro and Flash
`auto-gemini-2.5`	Auto	Routes between Gemini 2.5 Pro and Flash
`auto`	Auto	Routes to the latest available models

Selecting a Model

bash

```
# CLI flag
gemini --model gemini-3-pro-preview

# Environment variable
export GEMINI_MODEL=gemini-2.5-pro

# Settings file
{
  "model": {
    "name": "gemini-3-pro-preview"
  }
}
```

Auto Model Routing

When using an `auto` model, Gemini CLI uses a classifier to route each request to the optimal model:

- **Simple tasks** (single file edits, quick lookups) route to **Flash** for speed
- **Complex tasks** (multi-step debugging, architecture, strategic planning) route to **Pro** for quality
- **Plan mode** always routes to **Pro**; approved plan implementation routes to **Flash**

The classifier evaluates: number of steps required, ambiguity level, need for strategic planning, and debugging depth.

Changing Models Mid-Session

Use the `/model` slash command:

```
/model gemini-3-pro-preview
```

Slash Commands

Slash commands control the session and manage features. Type `/` to see all available commands.

Built-In Commands

Command	Description
`/help`	Show help and available commands
`/about`	Display version, model, auth, and sandbox info
`/clear`	Clear conversation history
`/copy`	Copy the last response to clipboard
`/auth`	Switch authentication method
`/model`	Change the active model
`/plan`	Toggle Plan/Act mode
`/memory`	Manage memory (show, add, refresh, list, inbox)
`/init`	Generate a GEMINI.md file for the project
`/restore`	Restore from a session checkpoint
`/editor`	Open an external editor for multi-line input
`/directory`	Browse and switch working directories
`/docs`	Open Gemini CLI documentation
`/bug`	Report a bug
`/corgi`	A surprise
`/chat`	Manage saved prompts

Extension Management

Command	Description
`/extensions list`	List installed extensions
`/extensions install <name>`	Install an extension
`/extensions uninstall <name>`	Remove an extension
`/extensions update <name>`	Update an extension
`/extensions enable <name>`	Enable a disabled extension
`/extensions disable <name>`	Disable an extension
`/extensions link <path>`	Link a local extension for development
`/extensions explore`	Browse available extensions

`/extensions configure <name>`	Configure extension settings

MCP Server Management

Command	Description
`/mcp list`	List configured MCP servers
`/mcp add`	Add an MCP server
`/mcp remove <name>`	Remove an MCP server
`/mcp enable <name>`	Enable a server
`/mcp disable <name>`	Disable a server

Hook Management

Command	Description
`/hooks`	Show all configured hooks
`/hooks enable <name>`	Enable a hook
`/hooks disable <name>`	Disable a hook
`/hooks migrate`	Migrate hooks from legacy format

Agent Management

Command	Description
`/agents`	List registered agents
`/agents enable <name>`	Enable an agent
`/agents disable <name>`	Disable an agent
`/agents reload`	Reload the agent registry

Skills Management

Command	Description
`/skills list`	List available skills
`/skills install <name>`	Install a skill
`/skills enable <name>`	Enable a skill

`/skills disable <name>`	Disable a skill
`/skills uninstall <name>`	Remove a skill

Memory System

Gemini CLI has a persistent memory system that remembers context across sessions.

How It Works

Memory is stored in `GEMINI.md` files. The model reads these at the start of every session and can update them during conversations.

Viewing Memory

```
/memory show
```

Shows the current memory contents from all applicable GEMINI.md files.

Adding Memory

```
/memory add Always use TypeScript strict mode in this project
```

Or let the model save memories naturally during conversation -- it uses the `save_memory` tool when it learns something worth remembering.

Managing Memory Files

```
/memory list          # List all GEMINI.md files in scope
/memory refresh       # Re-read memory files
/memory inbox         # View skill suggestions and patches
```

Memory Hierarchy

Memory files cascade:

1. **Global**: `~/.gemini/GEMINI.md` -- applies to all projects
2. **Project**: `./GEMINI.md` -- applies to this project
3. **Directory**: `./src/GEMINI.md` -- applies within `src/`

More specific memories override global ones for the same topic.

Sessions

Resuming Sessions

Continue where you left off:

bash

```
# Resume the most recent session
gemini --resume

# Resume a specific session
gemini --resume &lt;session-id&gt;
```

Listing Sessions

bash

```
gemini --list-sessions
```

Deleting Sessions

bash

```
gemini --delete-session &lt;session-id&gt;
```

Session Retention

Sessions are automatically cleaned up based on your retention settings:

json

```
{
  "general": {
    "sessionRetention": {
      "enabled": true,
      "maxAge": "30d",
      "maxCount": 100,
      "minRetention": "1d"
    }
  }
}
```

Checkpointing

Enable git-based checkpoints to snapshot your project state at key moments:

json

```
{
  "general": {
    "checkpointing": {
      "enabled": true
    }
  }
}
```

Restore from a checkpoint:

```
/restore
```

This presents available checkpoints and can restore both conversation history and git state.

Working with Files

File References with @

Reference files directly in your prompts:

```
Explain @src/auth/middleware.ts
```

```
Compare @package.json with @package-lock.json
```

> **Tip**
> If you paste text containing `@` symbols (like email addresses), you can enable `ui.escapePastedAtSymbols` in settings to prevent unintended file references.

Including Directories

Use `--include-directories` to specify additional directories the model should have access to:

bash

```
gemini --include-directories /path/to/shared/libs
```

Worktree Mode

For risky or exploratory work, use worktree mode to work in an isolated git branch:

bash

```
gemini --worktree
```

This creates a temporary git worktree so changes don't affect your main branch until you're ready to merge.

Sandbox

Sandboxing isolates shell command execution to protect your system from unintended changes.

Sandbox Modes

Mode	Platform	Description
Docker	Linux, macOS, Windows	Runs commands in a Docker container
Podman	Linux	Runs commands in a Podman container
macOS sandbox-exec	macOS	Uses macOS native sandboxing with seatbelt profiles
None	All	No sandboxing (default in some configurations)

Enabling Sandbox

bash

```
# Docker sandbox
export GEMINI_SANDBOX=docker
```

```
# Custom Docker image
export GEMINI_SANDBOX_IMAGE=my-dev-image:latest

# Via CLI flag
gemini --sandbox docker
```

Safe Command Allowlist

Even in sandbox mode, known read-only commands execute without confirmation:

- File viewing: `cat`, `head`, `tail`, `ls`, `wc`, `stat`
- Search: `grep`, `rg`, `find` (without `-exec` or `-delete`)
- Git read-only: `git status`, `git log`, `git diff`, `git show`, `git branch` (list only)
- System info: `pwd`, `whoami`, `uname`, `which`

Commands outside this list require explicit approval (unless in YOLO mode).

Proxy in Sandbox

If you're behind a corporate proxy:

bash

```
export GEMINI_SANDBOX_PROXY_COMMAND="your-proxy-setup-command"
```

The `HTTPS_PROXY`, `HTTP_PROXY`, and `NO_PROXY` environment variables are also respected.

Extensions

Extensions add capabilities to Gemini CLI through MCP servers, custom commands, hooks, policies, themes, and skills.

Extension Manifest

Extensions are defined by a `gemini-extension.json` file:

json

```
{
  "name": "my-extension",
  "version": "1.0.0",
  "description": "Adds custom tools",
  "mcpServers": {
    "my-server": {
      "command": "node",
      "args": ["server.js"]
    }
  },
  "skills": {
    "my-skill": {
      "path": "./skills/my-skill"
    }
  }
}
```

Installing Extensions

bash

```
# From the registry
gemini extensions install my-extension

# Or via slash command
/extensions install my-extension
```

Creating Extensions

bash

```
gemini extensions new my-extension
```

This scaffolds a new extension with the required structure.

Extension Components

Component	Directory	Description
MCP Servers	defined in manifest	External tool providers
Commands	`.gemini/commands/`	Custom TOML-based slash commands
Skills	`.gemini/skills/`	Reusable prompt templates (SKILL.md)
Hooks	defined in settings	Lifecycle event handlers
Policies	`.gemini/policies/`	Tool execution rules
Themes	defined in settings	UI color customization
Agents	`.gemini/agents/`	Custom agent definitions (.md files)

MCP Servers

Model Context Protocol (MCP) servers provide external tools and resources to Gemini CLI.

Configuration

Add MCP servers to your `settings.json`:

json

```
{
  "mcpServers": {
    "my-database": {
      "command": "node",
      "args": ["./mcp-servers/database.js"],
      "env": {
        "DB_HOST": "localhost"
      },
      "timeout": 30000
```

```
    }
  }
}
```

Managing Servers

bash

```
# Add interactively
gemini mcp add

# List servers
gemini mcp list

# Remove a server
gemini mcp remove my-database

# Enable/disable
gemini mcp enable my-database
gemini mcp disable my-database
```

Restricting MCP Servers

Limit which MCP servers can be loaded:

bash

```
gemini --allowed-mcp-server-names my-database,my-api
```

Hooks

Hooks let you run custom scripts at key points in the Gemini CLI lifecycle.

Hook Events

Event	Fires When
`BeforeTool`	Before any tool executes
`AfterTool`	After a tool completes
`BeforeModel`	Before sending a request to the model
`AfterModel`	After receiving a model response
`BeforeToolSelection`	Before the model chooses which tools to use

Defining Hooks

Add hooks to your `settings.json`:

json

```
{
```

```
  "hooks": {
    "BeforeTool": [
      {
        "command": "node ./hooks/validate-tool.js",
        "timeout": 5000
      }
    ],
    "AfterTool": [
      {
        "command": "node ./hooks/log-tool.js",
        "timeout": 5000
      }
    ]
  }
}
```

Hook Decisions

Hooks can influence tool execution by returning JSON:

json

```
{
  "decision": "allow",
  "reason": "Tool approved by policy"
}
```

json

```
{
  "decision": "block",
  "reason": "This tool is not allowed on production files"
}
```

json

```
{
  "decision": "allow",
  "continue": false,
  "stopReason": "All tasks completed"
}
```

Hook Capabilities

- **Block tools**: Return `"decision": "block"` to prevent execution
- **Modify arguments**: Return modified `toolInput` to alter tool parameters
- **Add context**: Return `additionalContext` to inject information into the model's context
- **Stop execution**: Return `"continue": false` to halt the agent loop
- **Control tool selection**: `BeforeToolSelection` hooks can set tool calling mode to `NONE`, `ANY`, or `AUTO`

> **Note**
>
> Hooks execute with a configurable timeout (default: 30 seconds). If a hook times out, it's treated as a failure. Hook errors do not automatically block tool execution unless the hook explicitly returns a block decision.

Policies

Policies define rules for tool execution approval.

Policy Files

Create policy files to automate approval decisions:

yaml

```yaml
# .gemini/policies/auto-approve-reads.yaml
rules:
  - tool: read_file
    decision: allow
  - tool: list_directory
    decision: allow
  - tool: run_shell_command
    args_pattern: "git status.*"
    decision: allow
```

Policy Paths

bash

```bash
# CLI flag
gemini --policy ./my-policies/

# Admin policies (cannot be overridden by workspace settings)
gemini --admin-policy ./admin-policies/
```

Settings Configuration

json

```json
{
  "policyPaths": ["./policies/"],
  "adminPolicyPaths": ["/org/policies/"]
}
```

Themes

Customize the look and feel of the CLI.

Built-In Themes

Set via settings:

json

```json
{
  "ui": {
    "theme": "Dark"
  }
}
```

Auto Theme Switching

Gemini CLI detects your terminal background color and switches between light and dark themes:

```json
{
  "ui": {
    "autoThemeSwitching": true,
    "terminalBackgroundPollingInterval": 60
  }
}
```

Custom Themes

Define custom themes in settings:

```json
{
  "ui": {
    "customThemes": {
      "my-theme": {
        "colors": {
          "primary": "#00ff00",
          "background": "#1a1a1a"
        }
      }
    },
    "theme": "my-theme"
  }
}
```

Extension Themes

Themes can also be distributed as part of extensions. See the `shades-of-green` example extension for a reference implementation.

Common Tasks

Code Review

```
Review the changes in the last commit for bugs and style issues
```

```bash
git diff HEAD~3 | gemini -p "Review these changes"
```

Bug Fixing

```
The login page throws a 403 error after Google sign-in. Debug this.
```

Refactoring

```
Refactor the user service to use dependency injection instead of global state
```

Writing Tests

```
Write unit tests for src/utils/validation.ts with vitest
```

Documentation

```
Generate JSDoc comments for all exported functions in src/api/
```

Project Setup

```
Set up a new Express API with TypeScript, ESLint, and Vitest
```

Git Operations

```
Create a new branch, make the changes we discussed, and prepare a commit message
```

Debugging

```
The test suite is failing with "Maximum update depth exceeded". Help me find the root cause.
```

Multi-File Edits

```
Rename the UserService class to AccountService across the entire codebase
```

Shell Scripting

```
Write a bash script that backs up the database, compresses it, and uploads to S3
```

Workflow Recipes

Recipe 1: Quick Bug Fix with Plan-Then-Act

bash

```
gemini --approval-mode plan
```

```
Analyze the bug in issue #123. Create a plan to fix it.
```

Review the plan, then press `Tab` to switch to Act mode:

```
Implement the plan
```

Recipe 2: Automated Test Suite Expansion

bash

```
gemini --approval-mode auto_edit -p "Find all untested functions in src/ and write vitest tests for them"
```

Recipe 3: Code Migration

bash

```
gemini -pi "Migrate all class components in src/components/ to functional components with hooks"
```

Recipe 4: Security Audit

bash

```
gemini --approval-mode plan -p "Audit this project for OWASP Top 10 vulnerabilities and create a report"
```

Recipe 5: API Documentation Generation

bash

```
gemini -p "Generate OpenAPI 3.0 spec for all REST endpoints in src/routes/"
```

Recipe 6: Dependency Upgrade

bash

```
gemini -pi "Upgrade all dependencies to their latest major versions. Fix any breaking changes."
```

Recipe 7: Performance Profiling

```
Run the benchmark suite, identify the three slowest functions, and suggest optimizations
```

Recipe 8: CI/CD Pipeline Setup

bash

```
gemini -p "Create a GitHub Actions workflow that runs tests, linting, and builds on PR"
```

Recipe 9: Database Migration

```
Generate a migration script to add the 'preferences' column to the users table
```

Recipe 10: Monorepo Task

bash

```
gemini --include-directories packages/shared -p "Add the new validation utility to the shared package and up
```

Recipe 11: Log Analysis

bash

```
cat production.log | gemini -p "Find error patterns and suggest fixes"
```

Recipe 12: Interactive Debugging Session

bash

```
gemini

&gt; The API returns 500 on POST /users. Let's debug this step by step.
&gt; [Model investigates, you guide with follow-ups]
&gt; Now check if the database connection pool is configured correctly
&gt; [Continue iterating until resolved]
```

Recipe 13: Code Review Workflow

bash

```
git diff main..feature-branch | gemini -p "Review this PR. Focus on security, performance, and maintainabili
```

Recipe 14: Scaffolding a New Module

```
Create a new authentication module with:
- JWT token generation and validation
- Refresh token rotation
- Rate limiting middleware
- Unit tests for all functions
```

Recipe 15: Working with Worktrees

bash

```
gemini --worktree -p "Experiment with converting the auth system to use Passport.js"
```

Changes happen in an isolated branch. Review and merge only if satisfied.

Recipe 16: Multi-Step Automation

bash

```
gemini --yolo --sandbox docker -p "Clone the upstream repo, apply our patches, run tests, and report results
```

Recipe 17: Extension Development

bash

```
gemini extensions new my-tool
cd my-tool
gemini -pi "Add an MCP server that provides a 'query_database' tool for PostgreSQL"
```

Recipe 18: Non-Interactive Scripting

bash

```
#!/bin/bash
RESULT=$(gemini --output-format json -p "List all TODO comments in src/")
echo "$RESULT" | jq '.items[] | .file + ":" + .line'
```

Recipe 19: Context-Rich Analysis

```
@src/auth/ @src/middleware/ @docs/security.md
Analyze the authentication flow end-to-end and identify any gaps
```

Recipe 20: Checkpoint-Based Exploration

Enable checkpointing, try risky changes, restore if needed:

json

```
{ "general": { "checkpointing": { "enabled": true } } }
```

```
Try converting the build system from webpack to vite
```

If it doesn't work:

```
/restore
```

Recipe 21: Batch Processing

bash

```
for dir in packages/*/; do
  gemini -p "Add missing TypeScript strict types to $dir" --approval-mode auto_edit
done
```

Recipe 22: Custom Command Workflow

Create `.gemini/commands/deploy.toml`:

toml

```
prompt = "Run the deployment checklist: lint, test, build, then deploy to staging"
```

Then use it:

```
/deploy
```

Recipe 23: Skill-Based Workflow

```
/skills list
```

Activate a skill for specialized behavior during the session.

Recipe 24: Web Research Integration

```
Search the web for the latest best practices on React Server Components and summarize them
```

Gemini CLI uses Google Search and URL fetch tools to pull current information.

Troubleshooting

Authentication Issues

Problem: `403 API Error` after Google sign-in

This commonly occurs when your Google account doesn't have access to the Gemini API. Solutions:

1. Verify your account has Gemini API access
2. Try a different auth method: `export GEMINI_API_KEY="your-key"`
3. Clear cached credentials and re-authenticate: `/auth`

Problem: `GEMINI_API_KEY` not recognized

bash

```
# Verify the variable is set
echo $GEMINI_API_KEY
```

```
# Ensure no trailing whitespace
export GEMINI_API_KEY="$(echo -n 'your-key')"
```

Problem: Vertex AI authentication fails

Ensure both required variables are set:

bash

```
export GOOGLE_CLOUD_PROJECT="your-project"
export GOOGLE_CLOUD_LOCATION="us-central1"
```

Or for express mode, just:

bash

```
export GOOGLE_API_KEY="your-key"
```

Installation Issues

Problem: `Cannot find module 'color-convert'`

This occurs in some environments (notably Google Colab). Reinstall:

bash

```
npm install -g @google/gemini-cli --force
```

Problem: Node version too old

bash

```
node --version  # Must be 20+
nvm install 20
nvm use 20
```

Startup Issues

Problem: Letter 'm' printed at startup

This is a known terminal escape code issue. It's cosmetic and doesn't affect functionality. Updating to the latest version resolves it.

Problem: Ripgrep download fails at startup

Gemini CLI bundles ripgrep for fast file search. If the download fails (network issues, corporate firewall), the CLI still works but file search may be slower. The fix:

1. Check network/proxy settings
2. Set `HTTPS_PROXY` if behind a corporate proxy
3. Restart the CLI after network issues are resolved

Problem: Slow startup

If the CLI takes a long time to start, it may be waiting for ripgrep to download. Set `HTTPS_PROXY` or ensure network access.

Session Issues

Problem: `Maximum update depth exceeded`

This is a React rendering loop in the TUI. Usually caused by:

1. Very large tool outputs overwhelming the renderer
2. Rapidly repeated tool calls

Solutions:

- Enable `model.summarizeToolOutput` for large outputs
- Increase `model.compressionThreshold` to compress context earlier
- Start a new session: `gemini`

Problem: Session won't resume

bash

```
# List available sessions
gemini --list-sessions

# Delete corrupted session
gemini --delete-session &lt;id&gt;
```

Model Issues

Problem: Quota exceeded

Gemini CLI handles quota errors automatically with retries and fallback model chains. If you consistently hit limits:

1. Switch to a lighter model: `/model gemini-2.5-flash`
2. Configure billing overage strategy:

json

```
{
  "billing": {
    "overageStrategy": "ask"
  }
}
```

Problem: Model not found

Ensure you're using a valid model name. Check available models with `/model`.

Problem: Model loops (repeats the same action)

Gemini CLI has built-in loop detection. If it triggers repeatedly:

1. Start a new session
2. Rephrase your request more specifically
3. Break complex tasks into smaller steps

Sandbox Issues

Problem: Docker sandbox won't start

bash

```
# Verify Docker is running
docker ps
```

```
# Check the sandbox image
docker pull &lt;image-name&gt;

# Try without sandbox temporarily
GEMINI_SANDBOX="" gemini
```

Problem: Sandbox can't access network

Configure proxy settings for the sandbox:

bash

```
export GEMINI_SANDBOX_PROXY_COMMAND="setup-proxy.sh"
```

Extension Issues

Problem: Extension not loading

bash

```
# Validate the extension
gemini extensions validate

# Check for errors
gemini extensions list
```

Problem: MCP server fails to start

Check the server command and arguments in your settings. Verify the server binary exists and is executable.

Hook Issues

Problem: Hook timeout

Increase the timeout in your hook configuration:

json

```
{
  "hooks": {
    "BeforeTool": [
      {
        "command": "my-hook.sh",
        "timeout": 60000
      }
    ]
  }
}
```

Problem: Hook blocks unexpectedly

Check your hook's output. A `"decision": "block"` return will prevent tool execution. Debug by running the hook command manually.

Performance Issues

Problem: High memory usage

- Lower `model.compressionThreshold` to compress context sooner

- Enable `model.summarizeToolOutput` for tools with large outputs
- Limit session length with `model.maxSessionTurns`

Problem: Slow responses

- Switch to a Flash model: `/model gemini-2.5-flash`
- Reduce context by starting a new session
- Ensure you're not behind a slow proxy

Debug Logging

Enable detailed debug output:

bash

```
DEBUG=* gemini -p "test"

# Or write to a file
export GEMINI_DEBUG_LOG_FILE=/tmp/gemini-debug.log
gemini
```

FAQ

Q: Is Gemini CLI free?

A: Gemini CLI is free to use with a Google account (Sign in with Google). Usage is subject to Gemini API quotas. Enterprise features through Vertex AI may incur charges.

Q: What models does Gemini CLI support?

A: Gemini 3.x (Pro, Flash), Gemini 2.5 (Pro, Flash, Flash-Lite), and auto-routing modes. The default auto mode selects the best model for each task.

Q: Can I use Gemini CLI offline?

A: No. Gemini CLI requires internet access to communicate with the Gemini API. All model inference happens server-side.

Q: Does Gemini CLI read my entire codebase?

A: Gemini CLI reads files as needed to answer your questions and complete tasks. It respects `.gitignore` by default (configurable via `fileFiltering.respectGitIgnore`). It does not upload your entire codebase.

Q: How do I stop Gemini CLI from modifying files?

A: Use Plan mode (`--approval-mode plan`) for read-only analysis. In default mode, every file modification requires your explicit approval.

Q: Can I use Gemini CLI in CI/CD pipelines?

A: Yes. Use non-interactive mode with `-p`, set `GEMINI_API_KEY` for auth, and use `--output-format json` for machine-readable output. Consider `--raw-output` for direct text output.

Q: How does the sandbox work?

A: The sandbox wraps shell command execution in a container (Docker or Podman) or OS-level sandbox (macOS). Your project files are mounted read-write inside the container. Network access is configurable.

Q: What is GEMINI.md?

A: A markdown file in your project that provides persistent context to Gemini CLI. It's like a project brief that the model reads at the start of every session. Use `/init` to generate one automatically.

Q: Can I restrict which tools Gemini CLI uses?

A: Yes. Use `--allowed-tools` to limit available tools, or create policies to control tool execution. Admin policies cannot be overridden by workspace settings.

Q: How do I use Gemini CLI with VS Code?

A: Install the Gemini CLI VS Code companion extension. It provides IDE integration for diff management, file context, and MCP bridging. Enable IDE mode in settings: `"ide": { "enabled": true }`.

Q: Can multiple people use the same Gemini CLI configuration?

A: Yes. Commit your `.gemini/settings.json`, `GEMINI.md`, and policy files to version control. System-level settings can be distributed via `GEMINI_CLI_SYSTEM_SETTINGS_PATH`.

Q: What happens if I run out of quota?

A: Gemini CLI detects quota exhaustion and can switch to AI credits if available. Configure this with `billing.overageStrategy`: `"ask"` (prompt each time), `"always"` (auto-use credits), or `"never"`.

Q: How do I report a bug?

A: Use the `/bug` slash command, which opens a browser to the GitHub issues page with pre-filled context.

Q: Can I use custom MCP servers?

A: Yes. Add MCP server configurations to your `settings.json` under `mcpServers`. Servers can be any process that speaks the MCP protocol.

Q: What programming languages does Gemini CLI support?

A: Gemini CLI is language-agnostic. It reads, writes, and reasons about any programming language the Gemini models understand, which covers all major languages.

Q: How do sessions work?

A: Each interactive session maintains conversation history. Sessions are saved automatically and can be resumed with `--resume`. Old sessions are cleaned up based on your retention settings.

Q: Can Gemini CLI search the web?

A: Yes. Gemini CLI includes web search and URL fetch tools that use Google Search for current information.

Q: What is the extension system?

A: Extensions bundle MCP servers, commands, skills, hooks, policies, themes, and agents into installable packages. They extend Gemini CLI's capabilities without modifying the core tool.

Q: How does context compression work?

A: When conversation context approaches the model's limit, Gemini CLI compresses earlier messages using a summarization model. The threshold is configurable via `model.compressionThreshold` (default: 0.5, meaning compression triggers at 50% capacity).

Q: Can I use Gemini CLI with monorepos?

A: Yes. Use `--include-directories` to include additional packages, and reference files across packages with `@path/to/file`. GEMINI.md files in subdirectories provide package-specific context.

Q: What is trusted folder mode?

A: Gemini CLI checks whether a workspace folder is trusted before loading its configurations (commands, skills, agents, MCP servers, hooks). This prevents malicious project configurations from executing automatically. You'll be prompted to trust new folders on first visit.

Q: How do I configure proxy settings?

A: Set `HTTPS_PROXY`, `HTTP_PROXY`, and `NO_PROXY` environment variables. For sandbox environments, also set `GEMINI_SANDBOX_PROXY_COMMAND`.

Q: Can Gemini CLI handle binary files and images?

A: Gemini 3.x models support multimodal input. You can reference images and other binary files for analysis, though the CLI's primary strength is with text-based code and configuration.

Q: What data does Gemini CLI send to Google?

A: Conversation contents (prompts, tool calls, responses) are sent to the Gemini API. Usage statistics are collected by default (disable with `privacy.usageStatisticsEnabled: false`). File contents are only sent when explicitly referenced or when tools read them in response to your requests.

Q: How do I use Gemini CLI with a screen reader?

A: Enable screen reader mode for plain-text output: `gemini --screen-reader` or set `"ui": { "accessibility": { "screenReader": true } }`.

CLI Reference

Global Options

Flag	Short	Description
`--model <name>`	`-m`	Set the Gemini model
`--prompt <text>`	`-p`	Non-interactive single prompt
`--prompt-interactive <text>`	`-pi`	Run prompt then enter interactive mode
`--approval-mode <mode>`		Set approval mode (default, auto_edit, yolo, plan)
`--yolo`		Enable YOLO mode (auto-approve everything)
`--sandbox <type>`		Set sandbox mode (docker, podman)
`--worktree`		Run in an isolated git worktree
`--resume [id]`		Resume a previous session
`--list-sessions`		List available sessions

`--delete-session <id>`		Delete a session
`--include-directories <paths>`		Additional directories to include
`--output-format <fmt>`		Output format: text or json
`--raw-output`		Raw model output to stdout
`--accept-raw-output-risk`		Acknowledge raw output risks
`--screen-reader`		Enable screen reader mode
`--policy <path>`		Load policy files
`--admin-policy <path>`		Load admin policy files
`--extensions <paths>`		Extension directories
`--list-extensions`		List installed extensions
`--allowed-tools <list>`		Restrict available tools
`--allowed-mcp-server-names <list>`		Restrict MCP servers
`--fake-responses <path>`		Use recorded responses (testing)
`--record-responses <path>`		Record responses to file (testing)
`--acp` / `--experimental-acp`		Enable Agent Communication Protocol

Subcommands

bash

```
gemini extensions &lt;command&gt;    # Manage extensions
gemini hooks &lt;command&gt;         # Manage hooks
gemini mcp &lt;command&gt;           # Manage MCP servers
gemini skills &lt;command&gt;        # Manage skills
```

Accessibility

Gemini CLI includes features for screen reader users and accessible environments:

- **Screen reader mode**: `--screen-reader` renders output as plain text
- **Vim mode**: Enable vi keybindings with `general.vimMode`
- **Loading phrases**: Configurable status messages while the model works
- **Terminal notifications**: System-level notifications for action-required prompts (`general.enableNotifications`)
- **Compact output**: `ui.compactToolOutput` reduces visual noise
- **Alternate buffer**: `ui.useAlternateBuffer` preserves shell history

DocCompiler.ai *v3.1 | Upgraded source coverage | Generated 2026-05-07 | Source:* `google-gemini/gemini-cli` *@* `a809bc7` *A Cyber Panda Solutions LLC product. This document is intentionally comprehensive -- built to be the complete, source-verified reference an AI or human needs without consulting other sources.* ***Do not print:*** *the always-current version lives at doccompiler.ai/google-gemini/gemini-cli.*