

OpenAI Codex Training

AI-Powered Software Engineering Agent

Press Space for next page →

Contact Info

Ken Kousen Kousen IT, Inc.

- ken.kousen@kousenit.com
- http://www.kousenit.com
- http://kousenit.org (blog)
- Social Media:
  - @kenkousen (twitter)
  - @kenkousen@foojay.social (mastodon)
  - @kousenit.com (bluesky)
- *Tales from the jar side* (free newsletter)
  - https://kenkousen.substack.com
  - https://youtube.com/@talesfromthejarside

Course Overview

5-Hour Hands-On Workshop

- Installation and authentication strategies
- Terminal UI and navigation
- Codex app, IDE, and cloud workflows
- Sandbox modes and approval policies
- Real-world coding projects

Topics We'll Cover

- Advanced TOML configuration
- **Agent Skills** - Reusable workflows
- MCP services integration
- Memory with AGENTS.md
- Custom prompts and profiles
- Model selection and local OSS options

Prerequisites

- Command-line experience
- Basic programming knowledge
- Git familiarity
- Docker (for advanced exercises)

What is OpenAI Codex?

A Coding Agent Across CLI, App, IDE, and Cloud

Key Features

- Frontier OpenAI models plus local OSS options
- Built-in safety with sandbox modes
- Rich configuration system
- Model Context Protocol (MCP) integration

Advanced Capabilities

- Session persistence and resumption
- Custom prompts and profiles
- CI/CD compatible
- Headless execution

Authentication Options

ChatGPT Account (Recommended)

- Uses your ChatGPT account login
- Required for some Codex surfaces and model availability
- Simplified login flow

API Key

- Direct API access
- Pay-per-use pricing
- Some Codex app/cloud features and models may not be available

Model Support

- **GPT-6.1-Sol** - Current default; GPT-6 frontier model for agentic coding (272K context)
- **GPT-6-Astra / GPT-6-Sol** - Previous GPT-6 frontier pair, still available
- **GPT-6-Luna** - Fast scoped work and lighter sub-agent tasks
- **GPT-5.6** - Previous generation, still available
- Codex picks your account's default model — check it with `/model`
- Local OSS models through Ollama or LM Studio
- Use `/model` in the TUI — or `codex debug models` for the full catalog

Installation Methods

```
# Official install script (macOS/Linux)
curl -fsSL https://chatgpt.com/codex/install.sh | sh

# NPM
npm install -g @openai/codex

# Homebrew (macOS/Linux)
brew install --cask codex

# Direct binary download
# Visit: https://github.com/openai/codex/releases
```

Verify Installation

```
codex --version
```

Configuration Locations

- **Config**: `~/.codex/config.toml`
- **Prompts**: `~/.codex/prompts/`
- **Logs**: `~/.codex/log/`

ChatGPT Account Login

```
# Interactive login
codex login

# Device-auth login for remote servers
codex login --device-auth
```

API Key Authentication

```
# Set environment variable
export OPENAI_API_KEY="your-key"

# Store it with Codex
printenv OPENAI_API_KEY | codex login --with-api-key
```

Do not put API keys directly in `config.toml`.

Verify Authentication

```
codex "Hello, are you working?"
```

Starting Codex

```
# Interactive mode (default)
codex

# With initial prompt
codex "explain this codebase"

# Execute and exit mode
codex exec "generate a README"
```

Key Bindings

- `Enter` - Submit prompt
- Type while Codex is working to steer it mid-turn
- `Ctrl+C` - Cancel current operation
- `Ctrl+D` - Exit Codex
- `Tab` - Autocomplete
- `/` - Access slash commands

Slash Commands

- `/status` - Show session info & token usage (`/usage` for account analytics)
- `/diff` - Review all pending changes
- `/clear` - Clear screen or start fresh chat
- `/save` - Save current session
- `/help` - Show available commands
- `/model` - Inspect or switch active model
- `/config` - Search and change settings live (flags the ones needing a restart)
- `/fast` - Toggle fast/standard mode
- `/permissions` - Manage sandbox permissions
- `/review` - Review changes without editing
- `/mcp` - Inspect MCP servers (`/mcp login <name>` signs in without leaving the session)
- `/skills` - Inspect available skills
- `/plugins` - Inspect installed plugins

/diff Command - Review Changes

```
--- a/src/main.py
+++ b/src/main.py
@@ -10,7 +10,9 @@ def process_data(input_file):
-    data = json.load(f)
+    with open(input_file, 'r') as f:
+        data = json.load(f)
     return data

3 files changed, 47 insertions(+), 12 deletions(-)
```

Review line-by-line before approving!

/status Command

Shows comprehensive session information:

```
Current model: gpt-6.1-sol
Session ID: abc123
Token usage: 15,432 / 272,000
Cost estimate: $0.46
Time elapsed: 12m 34s
```

/review Command

Launch code review without modifying your working tree:

```
# Review uncommitted changes
/review

# Review with specific focus
/review Check for security vulnerabilities

# Review changes against a branch
/review Compare with main branch
```

Built-in Reviewers

- **Security** - OWASP patterns, injection risks
- **Performance** - N+1 queries, memory leaks
- **Style** - Naming conventions, code structure
- **Tests** - Coverage gaps, edge cases

Reviewers analyze diffs without executing code

Search Your Codebase

```
# Fast text search with ripgrep
rg "TODO"
rg "authenticate"
rg "database connection"
```

- Respects .gitignore and runs fast on large repos
- Pipe matches into Codex for follow-up analysis
- Keep the agent focused by sharing only relevant snippets
- Great starting point for exploration and debugging

Web Search Capabilities

```
# One-off run with live web search enabled
codex --search "latest Spring Boot validation guidance"
```

- `web_search = "cached"` uses Codex's maintained index
- `web_search = "live"` fetches current web data
- `web_search = "disabled"` keeps the session offline
- Prefer official docs for APIs, SDKs, and product behavior
- Use MCP documentation servers when they are available

Using Web Search

```
# In interactive mode, Codex can search the web
codex
> "Search official docs for the current React Hook Form API"
> "Find Python async/await best practices"
> "What changed in Spring Boot validation docs?"
```

- Ask for current sources explicitly when accuracy matters
- Use live search for unstable or recent information
- Use cached or disabled search for reproducible coding work

Image Inputs

Attach screenshots and design specs for visual context:

```
# From command line
codex -i screenshot.png "Explain this error"
codex -i mockup.png "Implement this design"
codex -i diagram.png "Generate code for this architecture"

# Multiple images
codex -i error.png -i logs.png "Debug this issue"
```

Image Input Use Cases

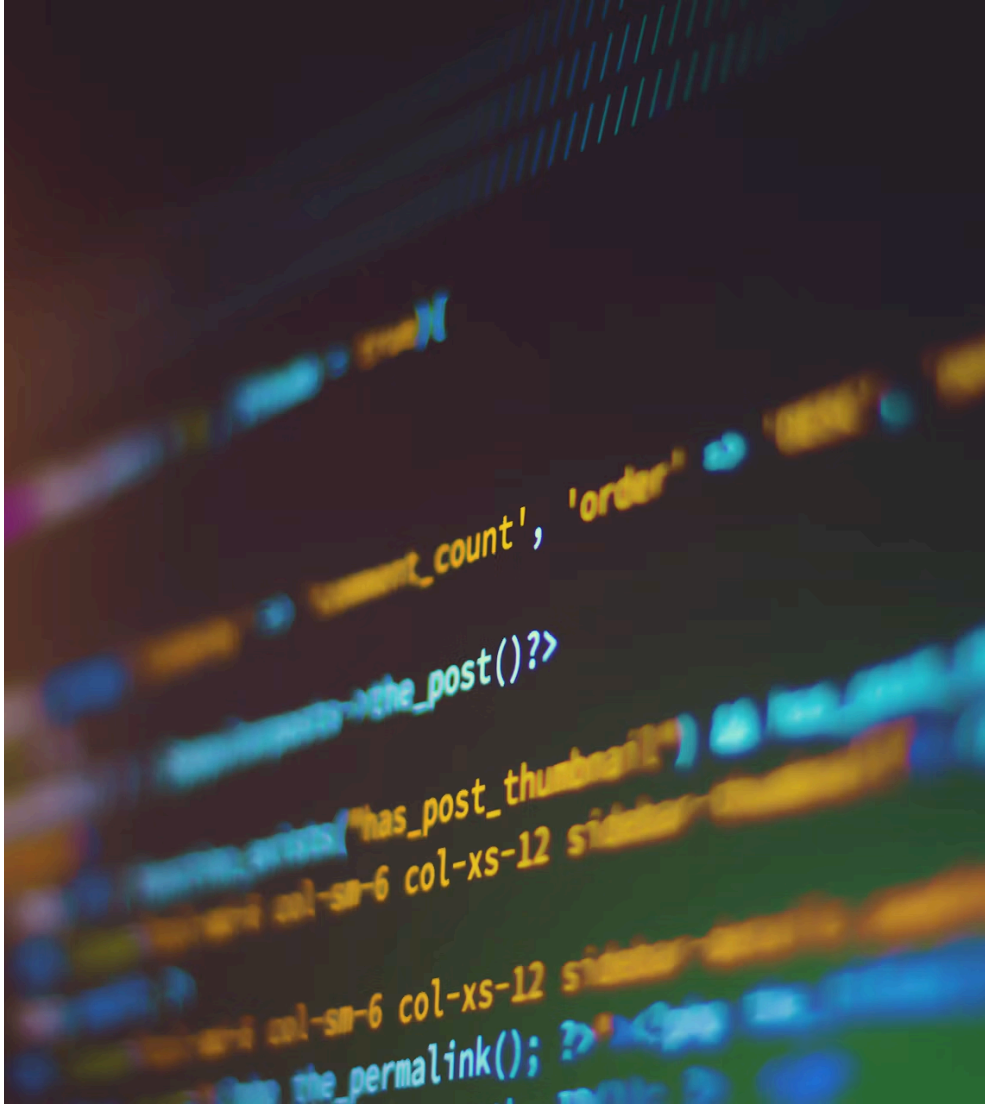
- **Debug UI errors** - Share error dialogs, stack traces
- **Implement designs** - Convert mockups to code
- **Analyze diagrams** - Generate from architecture docs
- **Review screenshots** - Identify accessibility issues
- **Compare outputs** - "Why does this look different?"

Paste images directly in the TUI composer!

Core Features

Essential Capabilities

Master the fundamentals



Sandbox Modes



- **read-only** - No file modifications
- **workspace-write** - Default; writes limited to the workspace
- **danger-full-access** - No sandboxing (use carefully!)

Approval Policies

- **on-request** - Approve risky actions (the interactive default)
- **never** - No approval prompts; best for narrow, non-interactive tasks
- `--approve-for-me` - Codex's automatic reviewer decides for you, inside the workspace-write sandbox
- Config also accepts **on-failure** (failures return straight to the model)
- `untrusted` was retired in 0.149 — configs still setting it refuse to load

Setting Safety Options

```
# Set sandbox mode
codex --sandbox read-only

# Set approval policy (equivalent to approval_policy in config)
codex --ask-for-approval on-request

# Full access only inside an external sandbox you trust
codex --sandbox danger-full-access --ask-for-approval on-request
```

CLI flags override config file settings for the session.

Project Memory: AGENTS.md

Automatic Context Loading

- Place `AGENTS.md` in project root
- Loaded automatically with first prompt
- Configurable size limit (default: 32KB)

Example AGENTS.md

```
# Project: E-Commerce Platform

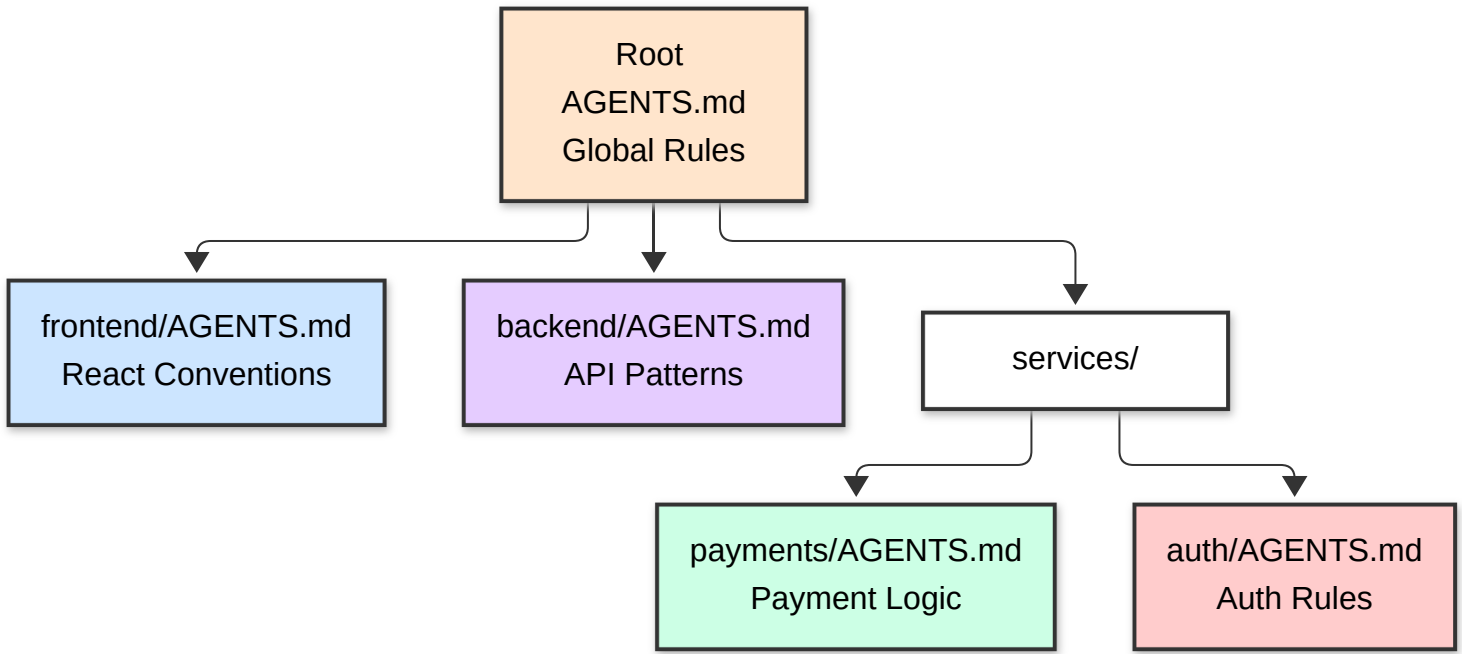
## Tech Stack
- Backend: Node.js + Express
- Database: PostgreSQL
- Frontend: React + TypeScript
```

AGENTS.md Best Practices

```
## Conventions
- Use async/await for all async operations
- Follow RESTful API patterns
- Write tests for all new features

## Current Focus
Working on payment integration with Stripe
```

Hierarchical AGENTS.md



Codex reads guidance from the project root down to the current directory. Closer files override earlier guidance when instructions conflict.

Context Cascade Benefits

- Global rules apply everywhere
- Subfolder rules override parent
- `AGENTS.override.md` takes precedence over `AGENTS.md` in the same directory
- Each team owns their conventions
- Frontend/backend stay independent
- Microservices maintain autonomy

Custom Prompts

Creating Custom Prompts

1. Create `.md` file in `~/.codex/prompts/`
2. Access via slash commands
3. Reusable across projects

Example Custom Prompt

```
# ~/.codex/prompts/refactor.md
Refactor the selected code following these principles:
1. Extract complex logic into small functions
2. Use meaningful variable names
3. Add appropriate error handling
```

Prompt Library Highlights

- Prebuilt prompts live in `~/.codex/prompts/` (see repo `prompts/README.md`)
- Core templates: `refactor`, `security-audit`, `test-gen`, `pr-review`, `api-upgrade`, `perf-fix`
- Customize or fork them for your team's workflow and slash commands

```
/refactor
/security-audit
/test-gen
/pr-review
/api-upgrade
/perf-fix
```

Prompt Arguments Workaround

- Custom prompts are static markdown files
- Solution: Use shell scripts as wrappers
- Scripts can accept parameters and build dynamic prompts
- Store in `~/.codex/scripts/` for reuse

Prompt Arguments: Implementation

```
#!/bin/bash
# ~/.codex/scripts/review-file.sh

FILE=$1
FOCUS=$2

cat > /tmp/review-prompt.md << EOF
Review the file ${FILE} focusing on ${FOCUS}:
- Check for bugs and errors
- Suggest improvements
- Rate code quality
EOF

codex exec "$(cat /tmp/review-prompt.md)"
```

Usage: `./review-file.sh UserService.java security`

Agent Skills

Reusable Workflows

Instructions, scripts, references, and assets



What Are Agent Skills?

- **Reusable instruction bundles** with optional scripts and resources
- **Authoring format** for workflows Codex can discover and invoke
- **Progressive loading**: Only name/description loaded at startup
- **Two invocation modes**: Explicit (`$skill-name`) or implicit (auto-detect)
- **Plugins** are the installable distribution unit for shared capabilities
- Managed from the CLI: `codex plugin add|list|remove` and `codex plugin marketplace add|list|upgrade`

Skill Locations

Scope	Location	Use Case
Repository	`.agents/skills/`	Team-shared skills
User	`~/.agents/skills/`	Personal workflows
Admin	System-managed	Enterprise policies
System	Bundled with Codex	Built-in workflows

Repo skills are discovered from the current directory up to the repo root.

Skill Structure

```
my-skill/
├── SKILL.md          # Required: YAML frontmatter + instructions
├── agents/
│   └── openai.yaml   # Optional: metadata, deps, invocation policy
├── scripts/          # Optional: executable code
├── references/       # Optional: documentation
└── assets/           # Optional: templates, resources
```

SKILL.md Format

```
---
name: security-review
description: >
  Perform security analysis. Use when asked about vulnerabilities,
  security audit, or "is this code secure".
---

# Security Review

Analyze codebases for security vulnerabilities...

## Workflow
1. Reconnaissance - identify tech stack
2. Dependency analysis - check for CVEs
3. Code analysis - scan for patterns
4. Generate report - create SECURITY_REVIEW.md
```

Invoking Skills

Explicit Invocation

```
# Use $ prefix to invoke directly
$skill-creator Create a skill for commit messages
$create-plan Design a new authentication system
```

Implicit Invocation

```
# Codex auto-selects based on task match
"Review this code for security vulnerabilities"
# → Automatically invokes security-review skill if installed
```

Built-in Skills

- **$skill-creator** - Bootstrap new skills from description
- **$skill-installer** - Install skills from catalog
- Additional skills depend on your local installation and plugins

Install additional skills:

```
$skill-installer linear    # Linear integration
$skill-installer notion    # Notion integration
```

Creating a Skill

```
# Use the built-in skill creator
$skill-creator Create a skill that generates
conventional commit messages based on staged changes
```

Codex will:

1. Create the skill folder structure
2. Generate SKILL.md with appropriate metadata
3. Add workflow instructions
4. Suggest reference files if needed

Skills vs Prompts

Aspect	Custom Prompts	Agent Skills
Location	`~/.codex/prompts/`	`~/.agents/skills/`
Structure	Single `.md` file	Folder with resources
Invocation	Slash commands	`$skill-name` or auto
Resources	Text only	Scripts, templates, refs
Sharing	Copy files	Git-friendly folders

Recommendation: Use Skills for complex, multi-step workflows

Skill Discovery in Practice

- Keep the `description` specific and action-oriented
- Use `$skill-name` when you want to force a skill
- Let implicit invocation handle repeated workflow matches
- Put large examples in `references/`, not `SKILL.md`
- Put deterministic helpers in `scripts/`

Configuration Profiles

One File per Profile (since 0.134)

Each profile is its own file — `~/.codex/<name>.config.toml` — with **top-level keys**, layered over `config.toml`:

```toml
# ~/.codex/production.config.toml
model = "gpt-6.1-sol"
approval_policy = "on-request"
sandbox_mode = "workspace-write"
```

```toml
# ~/.codex/quick.config.toml
model = "gpt-6-luna"
approval_policy = "never"
sandbox_mode = "read-only"
```

⚠️ Old-style `[profiles.*]` tables inside `config.toml` make the CLI **refuse to start** — if you configured profiles before 0.134, move them out.

Using Profiles

```
codex --profile production
codex --profile quick

# Catch stale or misspelled config keys early
codex --strict-config
```

Inside a session, `/config` searches every setting and applies supported changes immediately.

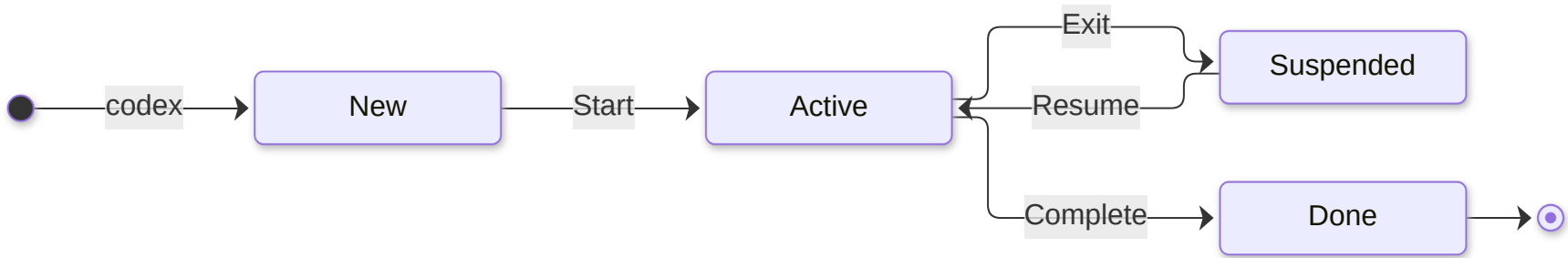
Resume Previous Sessions

```
# Open picker to choose a session
codex resume

# Resume the most recent session automatically
codex resume --last

# Resume a specific session by id
codex resume SESSION_ID
```

Session Lifecycle



Commands: `codex`, `codex resume`, `codex apply`

Session Commands

```
# Interactive session picker
codex resume

# Jump straight to the most recent session
codex resume --last

# Apply the last diff from the active session
codex apply
```

Non-Interactive Sessions

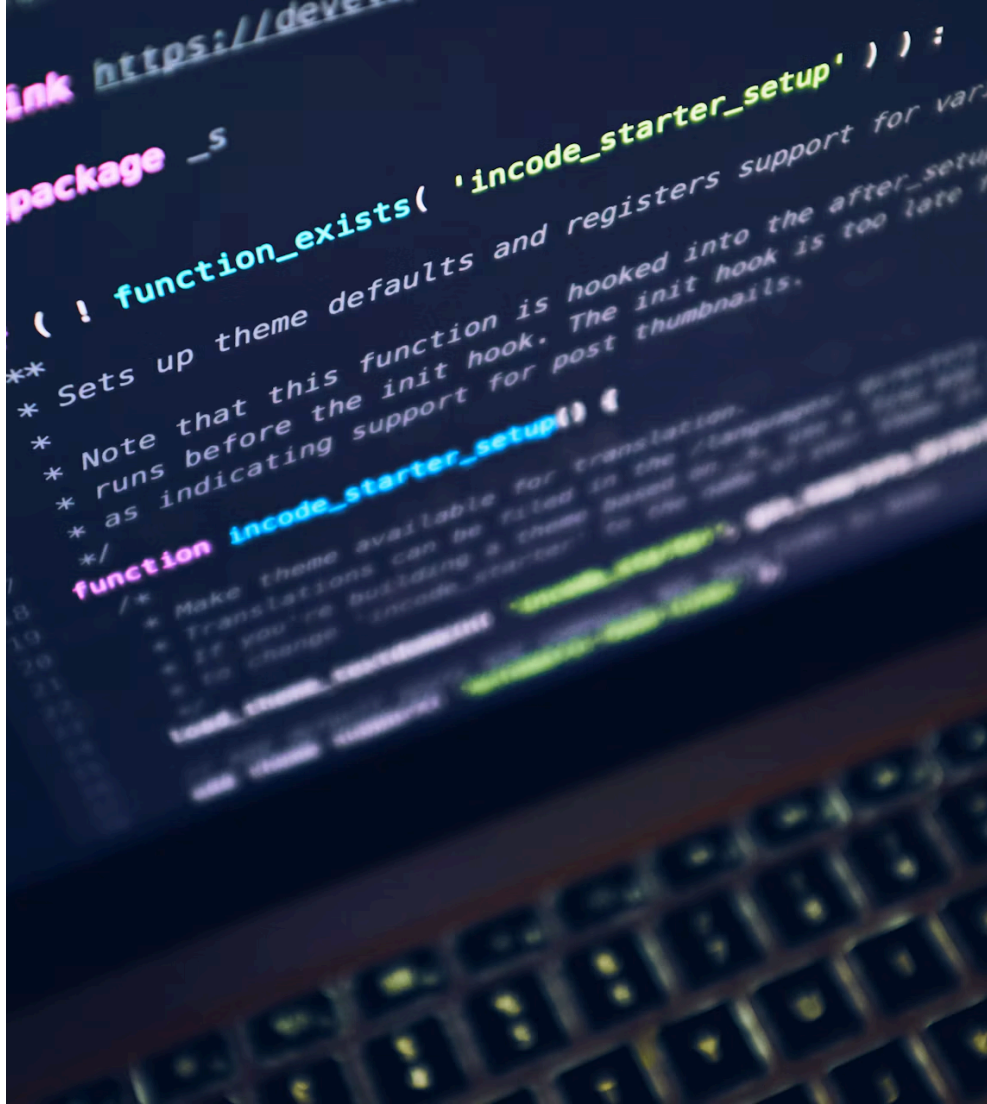
```
# Run in CI/CD pipeline
codex exec "update dependencies and fix breaking changes"

# Note: For resuming, use the regular command
codex resume
```

Advanced Features

Power User Tools

Unlock full potential



Codex Cloud

Run long tasks without tying up your terminal:

```
# Launch a cloud task
codex cloud exec "Refactor authentication module"

# Check task status
codex cloud status

# List running tasks
codex cloud list

# Inspect and apply results
codex cloud diff
codex cloud apply
```

Codex Cloud Benefits

- **Background execution** - Free up your terminal
- **Parallel tasks** - Run multiple jobs simultaneously
- **Persistent environments** - Pre-configured workspaces
- **Team collaboration** - Share environments and results
- **Long-running jobs** - Multi-hour refactoring sessions

IDE Extensions

Codex integrates with popular IDEs:

- **VS Code** - Full extension in marketplace
- **Cursor** - Native Codex support
- **Windsurf** - Integrated workflows

All extensions support:

- Skills and MCP servers
- Project-local configuration
- Same approval policies as CLI

Model Context Protocol

MCP Integration

Extend Codex with external tools and data



What Is MCP?

An open protocol that connects AI tools to external data sources and services.

- **Problem**: LLMs only know what's in their training data
- **Solution**: MCP lets them call external tools at runtime
- **Analogy**: Like USB for AI — a standard way to plug in capabilities
- **Adopted by**: Codex, Claude Code, Cursor, VS Code, and others

How MCP Works

MCP Concepts

Concept	Description
Server	A process that exposes tools (runs locally or remotely)
Tool	A function the server provides (e.g., "query-docs")
Transport	How Codex talks to the server: **stdio** (local) or **HTTP** (remote)
Discovery	On startup, Codex asks each server what tools it has
Invocation	During a conversation, Codex calls tools when relevant

Two Transport Types

stdio (Local Process)

Codex launches the server as a child process; they communicate over stdin/stdout.

```
[mcp_servers.context7]
command = "npx"
args = ["-y", "@upstash/context7-mcp"]
```

HTTP (Remote Server)

Codex connects to a running server over the network.

```
[mcp_servers.context7-remote]
url = "https://mcp.context7.com/mcp"
# If auth is required:
# bearer_token_env_var = "CONTEXT7_API_KEY"
```

Adding MCP Servers to Codex

CLI Command (Recommended)

```
codex mcp add context7 -- npx -y @upstash/context7-mcp
```

Verify

```
codex mcp list
```

Remove

```
codex mcp remove context7
```

Configuration in config.toml

The `codex mcp add` command writes to `~/.codex/config.toml`:

```
[mcp_servers.context7]
command = "npx"
args = ["-y", "@upstash/context7-mcp"]
```

Optional Settings

```
[mcp_servers.context7]
command = "npx"
args = ["-y", "@upstash/context7-mcp"]
startup_timeout_sec = 15   # Abort if server takes too long to start
tool_timeout_sec = 30      # Abort if a tool call takes too long
enabled = true             # Toggle without removing config
```

User-Level vs Project-Level

User-Level (default)

Available in all projects. Stored in `~/.codex/config.toml`.

```
codex mcp add context7 -- npx -y @upstash/context7-mcp
```

Project-Level

Only available in this project. Stored in `.codex/config.toml`.

```
[mcp_servers.context7]
command = "npx"
args = ["-y", "@upstash/context7-mcp"]
startup_timeout_sec = 20
tool_timeout_sec = 60
```

Project-level is useful when a server is specific to one codebase (e.g., a database server for that project's schema).

Context7: Live Documentation

Context7 is an MCP server that fetches up-to-date library documentation.

- **Two tools**: `resolve-library-id` and `query-docs`
- **Thousands of libraries**: React, Spring Boot, Zod, pytest, and more
- **Authentication optional in many setups**: Use environment variables if your setup requires a key
- **Why it matters**: Codex uses current APIs instead of stale training data

Context7 in Action

Without Context7

```
Create a Zod schema for a registration form
```

Codex relies on training data — might use outdated syntax.

With Context7

```
Use context7 to look up the current Zod docs, then create
a registration form schema with email and password validation
```

Codex fetches live Zod documentation first, then writes code using the current API.

More MCP Server Examples

GitHub

```
[mcp_servers.github]
command = "npx"
args = ["@modelcontextprotocol/server-github"]
env = { GITHUB_TOKEN = "${GITHUB_TOKEN}" }
```

PostgreSQL

```
[mcp_servers.postgres]
command = "npx"
args = ["@modelcontextprotocol/server-postgres"]
env = { CONNECTION_STRING = "${DATABASE_URL}" }
```

MCP Startup Guardrails

Prevent flaky servers from freezing Codex:

```
[mcp_servers.github]
command = "npx"
args = ["@modelcontextprotocol/server-github"]
startup_timeout_sec = 15  # Abort after 15 seconds
```

- Clean abort when servers fail to boot
- Prevents entire session from freezing
- Clear error messages on configuration issues

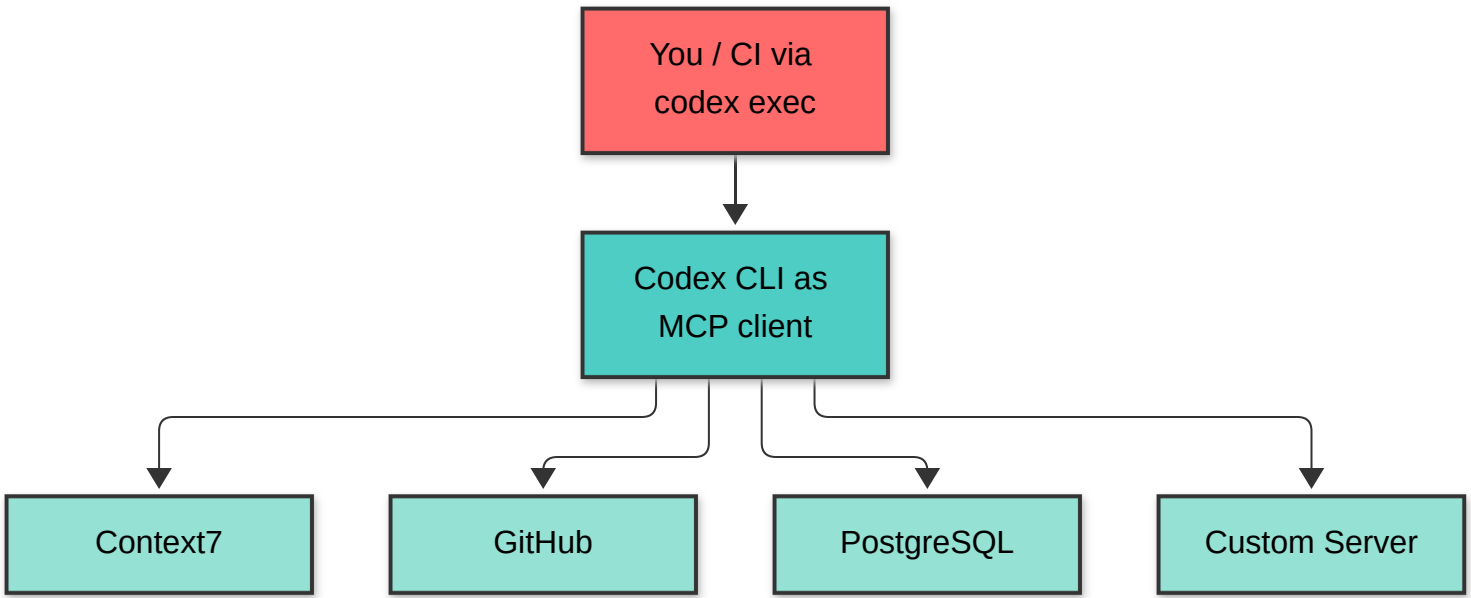
Calling Codex from Other Tools

`codex mcp-server` was removed in 0.154 — other tools now run Codex **headless**:

```
codex exec --sandbox read-only "Explain the failing test in src/"
codex exec --json -o result.md "Summarize this repo's build"
codex review --base main       # Non-interactive code review
```

- CI/CD pipelines call `codex exec` directly
- Other agents (e.g. Claude Code) shell out to `codex exec` / `codex review`
- Codex still applies its sandbox and approval policies
- `--json` and `-o` give the caller machine-readable output

MCP Architecture



Model Flexibility

Hosted and Local Options

OpenAI models, local OSS, and profiles



Model Profiles

Match the model to the work

Recommended OpenAI Profiles

```toml
# ~/.codex/quick.config.toml
model = "gpt-6-luna"
model_reasoning_effort = "low"
```

```toml
# ~/.codex/standard.config.toml
model = "gpt-6.1-sol"
model_reasoning_effort = "medium"
```

```toml
# ~/.codex/thorough.config.toml
model = "gpt-6.1-sol"
model_reasoning_effort = "high"
```

Local OSS Models

```
codex --oss --local-provider ollama
codex --oss --local-provider lmstudio
```

```
# ~/.codex/local.config.toml
model_provider = "ollama"
model = "qwen2.5-coder:32b"
sandbox_mode = "workspace-write"
approval_policy = "on-request"
```

Switch Models

```
codex --model gpt-6.1-sol
codex --model gpt-6-luna
codex --profile thorough

# What can my account use?
codex debug models
```

Enable Detailed Logging

```
# Set log level
export RUST_LOG=debug
codex

# Trace level (maximum detail)
export RUST_LOG=trace
codex
```

Log Locations

- Interactive: `~/.codex/log/codex-tui.log`
- Non-interactive: stderr output
- Custom: Redirect with shell operators

Debug Configuration

```
# Session-level debug output
RUST_LOG=debug codex

# Trace-level output for MCP startup issues
RUST_LOG=trace codex
```

CI/CD & Automation

- GitHub Actions example lives in repo (`.github/workflows/codex-review.yml`)
- Typical steps: checkout → install Codex → authenticate → `codex exec` → upload artifacts
- Cron ideas: weekly security sweep, dependency refresh, monthly cleanup
- Guardrails: run on branches, review PRs before merge, notify on failures

CI/CD Pipeline Examples

```
# Fail-fast pipeline
git pull && \
codex exec "migrate database schema" && \
npm test
```

```
# Weekly cron example
0 2 * * 1 cd /path/to/repo && \
  codex exec "weekly security audit"
```

```
# Chain commands to stop on failure
npm install && \
codex exec "fix any TypeScript errors" && \
npm run build
```

Advanced TOML Configuration

```toml
# ~/.codex/config.toml
model = "gpt-6.1-sol"
model_provider = "openai"
approval_policy = "on-request"
sandbox_mode = "workspace-write"
web_search = "cached"
model_reasoning_effort = "medium"
```

Environment & Notifications

```toml
notify = ["notify-send", "Codex", "Task completed"]

[shell_environment_policy]
inherit = "core"
exclude = ["OPENAI_API_KEY", "GITHUB_TOKEN"]

[shell_environment_policy.set]
NODE_ENV = "development"
```

Shell Environment Policies

- **all** - Inherit the parent shell environment
- **core** - Inherit the minimal useful shell environment
- **none** - Start from an empty environment, then add explicit variables

Environment Configuration

```
[shell_environment_policy]
inherit = "core"
exclude = ["AWS_SECRET_ACCESS_KEY", "OPENAI_API_KEY"]

[shell_environment_policy.set]
LANG = "en_US.UTF-8"
```

Explicit Environment Additions

```
[shell_environment_policy.set]
PATH = "/usr/local/bin:/usr/bin:/bin"
NODE_ENV = "development"
RUST_LOG = "warn"
```

Security Considerations

- Use `inherit = "none"` for tightly controlled automation
- Use `inherit = "core"` for most local development
- Never expose secrets in config

Notification Options

- Notifications use the top-level `notify = [...]` command
- Codex passes a JSON payload to the command on stdin
- Keep secrets in environment variables, not command arguments

Enterprise Features

- **MDM Configuration** - Managed settings on macOS via MDM profiles
- **Admin-scoped Skills** - Organization-wide skill deployment
- **Requirements.toml** - Enforce policies across teams
- **Zero Data Retention** - ZDR compliance with ChatGPT auth
- **Audit Logging** - Track all agent actions

Requirements.toml

Enforce organizational policies:

```toml
# /etc/codex/requirements.toml (UNIX)
# or via MDM (macOS)

[sandbox]
allowed_sandbox_modes = ["read-only", "workspace-write"]
# Prevents danger-full-access

[skills]
admin_only = ["deploy-prod", "db-migrate"]

[approval]
allowed_approval_policies = ["on-request"]

[web_search]
allowed_web_search_modes = ["disabled", "cached"]
```

Prompt Engineering Tips

When to Use Each Mode

Mode	Use Case
Interactive (`codex`)	Exploration, iteration, learning
Single prompt (`codex "..."`)	Quick questions, small tasks
Exec (`codex exec`)	Automation, CI/CD, scripts

Effective Prompts

Be specific about scope:

- ❌ "Fix the bugs"
- ✅ "Fix the null pointer exception in UserService.java line 42"

Provide context:

- ❌ "Add tests"
- ✅ "Add unit tests for the validateEmail function using Jest"

State expected outcomes:

- ❌ "Make it faster"
- ✅ "Optimize the database query to reduce response time below 100ms"

AGENTS.md Best Practices

Keep it focused and current:

```
# Project Context
E-commerce platform, Node.js + PostgreSQL

## Current Sprint
Payment integration with Stripe

## Conventions
- Use async/await, not callbacks
- All API responses follow { data, error } format
- Tests required for all new endpoints
```

Update AGENTS.md as your project evolves!



Practical Exercises

Hands-On Labs

Learn by doing

Exercise Structure

Three Main Categories

Available Labs

- **Lab 0**: Planning and Steering Warm-Up (15 min)
- **Lab 1**: Spring Boot REST API (generate from scratch)
- **Lab 2**: Python Refactoring (improve legacy code)
- **Lab 3**: React TypeScript Forms (frontend development)
- **Lab 4**: Microservices Architecture (multi-language)
- **Lab 5**: Skills Creation (extend Codex)
- **Lab 6**: MCP Servers with Context7 (connect external data)

Each Exercise Includes

- Starter code (where applicable)
- Step-by-step Codex prompts
- Success criteria checklist
- Advanced challenges

Note: You build the solution using Codex—no reference implementations provided!

Lab 0: Planning and Steering Warm-Up

- Objective: Practice planning, steering, permissions, and review commands
- Timebox: 15–20 minutes
- Workspace: `exercises/plan-mode-warmup`
- Fix bugs in a small Python module while learning the interaction model
- Practice `/status`, `/permissions`, `/diff`, `/model`, `/review`

Lab 1: Spring Boot API

- Objective: Build a Spring Boot 3 task-management REST API end-to-end
- Timebox: 60–90 minutes
- Workspace: `exercises/java-spring-boot`
- Instructions: open `exercises/java-spring-boot/README.md`

Lab 2: Python Refactoring

- Objective: Modernize legacy Python code with clean architecture and tests
- Timebox: 45–60 minutes
- Workspace: `exercises/python-refactoring`
- Instructions: open `exercises/python-refactoring/README.md`

Lab 3: React TypeScript Forms

- Objective: Ship a production-ready registration flow with React, TypeScript, and Zod
- Timebox: 45–60 minutes
- Workspace: `exercises/react-forms`
- Instructions: open `exercises/react-forms/README.md`

Lab 4: Microservices

- Objective: Build an event-driven multi-language microservices system
- Timebox: 90–120 minutes
- Workspace: `exercises/microservices`
- Instructions: open `exercises/microservices/README.md`

Lab 5: Skills Creation

- Objective: Create a custom skill using `$skill-creator`
- Timebox: 30 minutes
- Task: Build a skill that generates conventional commit messages
- Workspace: `exercises/skills-creation/`
- Instructions: open `exercises/skills-creation/README.md`

Lab 6: MCP Servers with Context7

- Objective: Configure an MCP server and use live documentation in prompts
- Timebox: 15–20 minutes
- Pre-req: Node.js 18+ and internet access
- Workspace: `exercises/mcp-context7/` (then apply to `exercises/react-forms/`)
- Instructions: open `exercises/mcp-context7/README.md`

Optional: Advanced Challenges

- **Database Migration**: Use Codex + MCP tools to modernize a legacy schema
- **AI Code Review**: Automate PR reviews using `.github/workflows/codex-review.yml`
- **Full-Stack Capstone**: Combine Labs 1-4 into a production-style application

These build on the core labs and can be explored as time permits.

Best Practices

Professional Workflows

Enterprise-ready patterns



Review Changes Before Approving

- Codex displays unified diffs automatically
- Use `/diff` to see all pending changes
- Review line-by-line for unintended edits
- Check file statistics (insertions/deletions)
- Catch mistakes before they land in codebase

Pro tip: Always review diffs for:

- Accidental deletions
- Unrelated file changes
- Security implications

Security Best Practices

Sandbox Configuration by Environment

Development Profile

```
# ~/.codex/dev.config.toml
sandbox_mode = "workspace-write"
approval_policy = "on-request"
```

Staging Profile

```
# ~/.codex/staging.config.toml
sandbox_mode = "workspace-write"
approval_policy = "on-request"
```

Production Profile

```
# ~/.codex/prod.config.toml
sandbox_mode = "read-only"
approval_policy = "on-request"
```

Security Guidelines

- Never store API keys in config files
- Use environment variables for secrets
- Use `read-only` for reviews and audits
- Enable approval for risky actions
- Regular audit of generated code
- Restrict network access in sandbox

Team Collaboration

Shared Configuration

Project AGENTS.md

```
# Team: Platform Engineering
## Conventions
- PR reviews required for all changes
- Follow company style guide
- Security scanning mandatory
- 80% test coverage minimum
```

Current Sprint Context

```
## Current Sprint
- Migrating to Kubernetes
- Implementing OAuth 2.0
```

Shared Prompts Repository

```
# Clone team prompts
git clone team-repo/codex-prompts ~/.codex/prompts

# Keep synchronized
cd ~/.codex/prompts && git pull
```

Model Selection Strategy

```
# ~/.codex/quick.config.toml
model = "gpt-6-luna"    # Fast scoped work
```

```
# ~/.codex/complex.config.toml
model = "gpt-6.1-sol"     # Complex reasoning
```

Local Models

```
# ~/.codex/local.config.toml
model_provider = "ollama"
model = "qwen2.5-coder:32b"  # Local model availability varies
```

Optimization Tips

- Use GPT-6-Luna for simple tasks
- Cache responses with session resumption
- Batch similar operations
- Use local models for sensitive data

Troubleshooting

Common Issues

Solutions and workarounds



Troubleshooting Guide

Common Issues & Solutions

Authentication Failures

```
# Clear cached credentials
rm -rf ~/.codex/auth

# Re-authenticate
codex login --device-auth
```

Sandbox Errors

```
# Confirm sandbox configuration
grep sandbox_mode ~/.codex/config.toml

# If the workspace is intentionally disposable
codex --sandbox danger-full-access --ask-for-approval on-request
```

MCP Connection Issues

```
# Test MCP server
npx @modelcontextprotocol/server-test

# Enable debug logging
RUST_LOG=trace codex
```

Context Limit Errors

Symptoms: "Context window exceeded", slow responses

Solutions:

- Start a new session: `/clear` or new terminal
- Use `codex resume` to continue with trimmed context
- Break large tasks into smaller prompts
- Remove verbose files from AGENTS.md

Model Availability Issues

```
# In the TUI, use /model to inspect or switch models

# Pick an explicit model from your catalog (codex debug models)
codex --model gpt-6.1-sol

# Verify API connectivity
curl -I https://api.openai.com/v1/models
```

Common Error Messages

Error	Cause	Fix
"Token expired"	Auth timeout	`codex login` again
"Rate limited"	Too many requests	Wait or use smaller model
"No such table"	Wrong osquery platform	Check platform docs
"Sandbox denied"	Permission blocked	Adjust sandbox mode

VS Code Integration

```
{
  "tasks": [{
    "label": "Codex Review",
    "type": "shell",
    "command": "codex exec 'Review ${file} for issues'"
  }]
}
```

Git Hooks

```
#!/bin/bash
# .git/hooks/pre-commit
codex exec -p quick -s read-only \
  "Check staged files for security issues"
```

(`codex exec` is the non-interactive entry point — there is no `-n` flag; `quick` is the read-only profile from the config examples.)

Make Integration

```
review:
  codex exec "Review all changes since last commit"

generate-tests:
  codex exec "Generate missing unit tests"
```

Diagnostics Workflow

- Reproduce with a narrow prompt
- Run with `RUST_LOG=debug` or `RUST_LOG=trace`
- Inspect `~/.codex/log/`
- Disable optional MCP servers if startup is flaky
- Capture command, model, sandbox, approval policy, and error text

Log Analysis

```
# Filter JSON logs for errors
cat ~/.codex/log/codex-tui.log | jq 'select(.level == "error")'

# Monitor in real-time
tail -f ~/.codex/log/codex-tui.log | grep ERROR
```

Build Your Own MCP Server

```
// custom-mcp-server.js
import { MCPServer } from '@modelcontextprotocol/sdk';

const server = new MCPServer({
  name: 'custom-tools',
  version: '1.0.0'
});
```

MCP Tool Definition

```
tools: [{
  name: 'database-query',
  description: 'Execute database queries',
  handler: async (params) => {
    return { result: 'Query executed' };
  }
}]
```

Register Custom Server

```toml
[mcp_servers.custom]
command = "node"
args = ["./custom-mcp-server.js"]
```

From GitHub Copilot

- Export commonly used snippets
- Convert to Codex prompts
- Leverage session persistence

From Other Agent Tools

- Move durable project guidance into `AGENTS.md`
- Convert reusable prompt text into `~/.codex/prompts/`
- Convert multi-step workflows into skills
- Recreate external integrations as MCP servers where possible

From Cursor/Codeium

- Migrate project context
- Recreate custom instructions
- Set up equivalent workflows

Current Codex Surfaces

- **CLI**: fastest way to work in a terminal or SSH session
- **IDE extensions**: editor-native context, diffs, and prompts
- **Codex App**: visual thread management, worktrees, and review
- **Codex Cloud**: background tasks on remote environments
- **MCP**: current docs, project data, services, and custom tools

Current Local Defaults

- Sandbox controls what Codex can technically access
- Approval policy controls when Codex asks you first
- Network is constrained unless explicitly enabled
- Project `.codex/config.toml` loads only for trusted projects
- `AGENTS.md` gives project and directory-specific guidance

Extension Points

- **AGENTS.md** for durable project guidance
- **Prompts** for reusable slash-command text
- **Skills** for multi-step workflows with resources
- **Plugins** for installable capabilities
- **MCP servers** for live tools and data
- **Hooks** for session events and local automation

Hooks

- Shell commands (or MCP tools) that fire on lifecycle events — stable, on by default
- **Events**: `PreToolUse`, `PostToolUse`, `SessionStart`, `SessionEnd`, `UserPromptSubmit`, `Notification`, `Stop`, `SubagentStop`
- Config: `~/.codex/hooks.json` or a `[hooks]` table in `config.toml`
- First run prompts you to **trust** the hook (`--dangerously-bypass-hook-trust` skips — don't)
- Async command hooks and MCP-tool hooks landed in 0.148/0.149
- Admins can restrict to managed hooks: `allow_managed_hooks_only`

```
{ "hooks": { "PreToolUse": [ { "type": "command",
    "command": "~/.codex/hooks/block-force-push.sh" } ] } }
```

MCP Robustness

Always add timeout to MCP servers:

```
[mcp_servers.your_server]
command = "your-command"
startup_timeout_sec = 15
tool_timeout_sec = 60
```

Advanced Model Control

Use high reasoning for complex, long-running tasks:

```
codex -m gpt-6.1-sol -c model_reasoning_effort='high'
```

Or configure in TOML:

```
model_reasoning_effort = "high"  # low/medium/high/xhigh/max/ultra (ultra also delegates to sub-agents)
```

High Reasoning Mode Benefits

- Multi-hour work sessions allowed
- Iterates tests until green
- Deep problem-solving capability
- Automatic retry on failures
- Best for complex refactoring

When to Use High Reasoning

- Large test suite fixes
- Complex architectural changes
- Multi-file refactoring
- Performance optimizations
- Breaking change migrations

Network Access Control

Control network access in sandbox mode:

```
# ~/.codex/config.toml
[sandbox_workspace_write]
network_access = true  # Default: false
```

Keep network off for reproducible local tasks. Enable it only when installs, integration tests, or live documentation require it.

Network Control Benefits

- Offline by default for reproducibility
- Allow only specific staging APIs
- Prevent accidental external calls
- Maintain test isolation
- Control data exfiltration

Network Allowlist Use Cases

- Integration tests with staging APIs
- CI/CD pipelines with controlled access
- Development with specific endpoints
- Security-sensitive environments
- Reproducible test suites

Codex App Workflows

- **Local mode**: work directly in the current project directory
- **Worktree mode**: isolate agent changes in a git worktree
- **Cloud mode**: run tasks in configured cloud environments
- **Visual review**: inspect diffs before landing changes
- **Computer use**: inspect apps and browsers when the task needs it

Computer Use Safety

- Keep the task narrow and stay present
- Close sensitive apps and tabs before granting access
- Review permission prompts carefully
- Be cautious with signed-in browser sessions
- Cancel if Codex focuses the wrong window or website

The Codex App

Standalone desktop application for visual Codex workflows

- Work in local projects, isolated worktrees, or cloud tasks
- Share MCP configuration and skills with the CLI and IDE
- Review diffs visually before accepting changes
- Useful for parallel threads and frontend/browser inspection

Codex App: Key Features

- **Multi-agent parallelism** — run multiple agents simultaneously, each in its own thread
- **Worktree isolation** — keep experimental changes separate
- **Visual diff review** — inspect and stage changes deliberately
- **Computer use** — inspect browser and desktop UI when needed
- **Cloud handoff** — run long tasks away from your terminal

CLI vs App: When to Use Which

Aspect	Codex CLI	Codex App
Interface	Terminal	Desktop GUI
Multi-project	Terminal tabs	Visual thread list
Diff review	Basic	Inline with comments
Computer use	Limited	Built-in app/browser control
Worktrees	`--worktree` flag or in-session tools	First-class workflow
Remote/SSH	Supported	Not supported
Speed	Faster	Slightly heavier

Use both: CLI for speed and remote work, App for visual management and parallel agents

Community & Ecosystem

- Open source at github.com/openai/codex
- Official docs at learn.chatgpt.com/docs (developers.openai.com/codex redirects)
- Skills, plugins, and MCP servers extend the base agent
- GitHub discussions and issues are the right place for project feedback

Essential Commands

```
# Basic usage
codex                          # Interactive mode
codex exec "prompt"            # Execute task & exit
codex resume                   # Resume session
codex resume --last            # Resume most recent session
codex apply                    # Apply last diff
codex update                   # Update Codex CLI
codex doctor                   # Diagnose install, config, auth, runtime
```

Configuration Commands

```
codex --profile production            # Use profile (~/.codex/production.config.toml)
codex --sandbox read-only             # Set sandbox
codex --ask-for-approval on-request   # Set approval (on-request | never)
codex --approve-for-me                # Automatic review approves on your behalf
codex --strict-config                 # Fail fast on stale/unknown config keys
```

Advanced Commands

```
codex mcp list                # List MCP servers
codex mcp login server-name   # OAuth login (or /mcp login <name> in a session)
codex cloud exec "prompt"     # Launch a cloud task
codex cloud diff              # Review cloud task changes
codex cloud apply             # Apply cloud task changes locally
codex review --base main      # Review your branch against main
codex exec --json -o out.md "…"  # Machine-readable run for CI
```

Key Files

- Config: `~/.codex/config.toml`
- Prompts: `~/.codex/prompts/*.md`
- Memory: `./AGENTS.md`
- Logs: `~/.codex/log/`

Documentation & Code

📚 Official Documentation

`https://learn.chatgpt.com/docs`

🐙 GitHub Repository

`https://github.com/openai/codex`

Course & Community

💻 Course Materials & Labs

`https://github.com/kousen/codex-training`

🆘 Community Support

`https://github.com/openai/codex/discussions`

📦 MCP Registry

`https://modelcontextprotocol.io/registry`

Contributing

- Bug reports: GitHub Issues
- Feature requests: GitHub Discussions
- Code contributions: Pull Requests

Ecosystem Examples

- MCP server templates
- Prompt libraries
- Configuration examples
- Integration guides

Choosing a Codex Surface

- **CLI** for speed, SSH, automation, and terminal-native workflows
- **IDE extension** for editor-local changes and inline context
- **App** for visual review, parallel threads, worktrees, and browser inspection
- **Cloud** for long-running background work
- **MCP** when the agent needs live docs, services, or custom tools

References & Credits

Course Sources

- Official Codex documentation
- OpenAI Codex GitHub repository
- Model Context Protocol documentation
- Course labs and configuration examples

Additional Resources

Official Sources

- Codex Documentation
- Codex CLI Documentation
- Codex App Documentation
- Codex MCP Documentation
- Agent Skills Guide
- Codex GitHub Repository
- Skills Catalog

Specifications & Community

- agentskills.io - Skills specification
- Model Context Protocol
- Codex Discussions

Thank You!