

OpenAI Codex CLI Training

AI-Powered Terminal Coding 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](https://twitter.com/kenkousen) (twitter)
 - [@kenkousen@foojay.social](https://mastodon.foojay.social/@kenkousen) (mastodon)
 - [@kousenit.com](https://bluesky.kousenit.com/@kousenit) (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
- Sandbox modes and approval policies
- Real-world coding projects

Topics We'll Cover

- Advanced TOML configuration
- MCP services integration
- Memory with AGENTS.md
- Custom prompts and profiles
- Multi-model provider support

Prerequisites

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

What is OpenAI Codex CLI?

Lightweight Terminal-Based Coding Agent

Key Features

- Multi-model support (GPT, Claude, Ollama)
- 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 existing ChatGPT subscription
- Zero Data Retention (ZDR)
- Simplified login flow

API Key

- Direct API access
- Pay-per-use pricing
- More configuration required

Model Support

- OpenAI GPT models (default)
- Anthropic Claude via API
- Local models via Ollama
- Custom providers via configuration

Installation Methods

```
# NPM (recommended)
npm install -g @openai/codex

# Homebrew (macOS/Linux)
brew install 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  
  
# Headless login for servers  
codex login --headless
```

API Key Authentication

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

# Or configure in TOML
echo 'api_key = "your-key"' >> ~/.codex/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
- `Ctrl+C` - Cancel current operation
- `Ctrl+D` - Exit Codex
- `Tab` - Autocomplete
- `/` - Access slash commands

Slash Commands

- `/status` - Show session info & token usage
- `/diff` - Review all pending changes
- `/clear` - Clear conversation history
- `/save` - Save current session
- `/help` - Show available commands
- `/settings` - Adjust runtime settings

/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 (v0.35+)

Shows comprehensive session information:

```
Current model: gpt-4
Session ID: abc123
Token usage: 15,432 / 128,000
Cost estimate: $0.46
Time elapsed: 12m 34s
```

Search Your Codebase

```
# Search for text in your project
codex --search "TODO"
codex --search "authenticate"
codex --search "database connection"
```

- Fast semantic search across all files
- Understands code context
- Shows relevant snippets
- Great for exploration and debugging

Web Search Capabilities

```
# ~/.codex/config.toml
[tools]
web_search = true
```

- Search the entire web for solutions
- Find latest documentation and tutorials
- Access Stack Overflow answers
- Get real-time information
- Research libraries and frameworks

Using Web Search

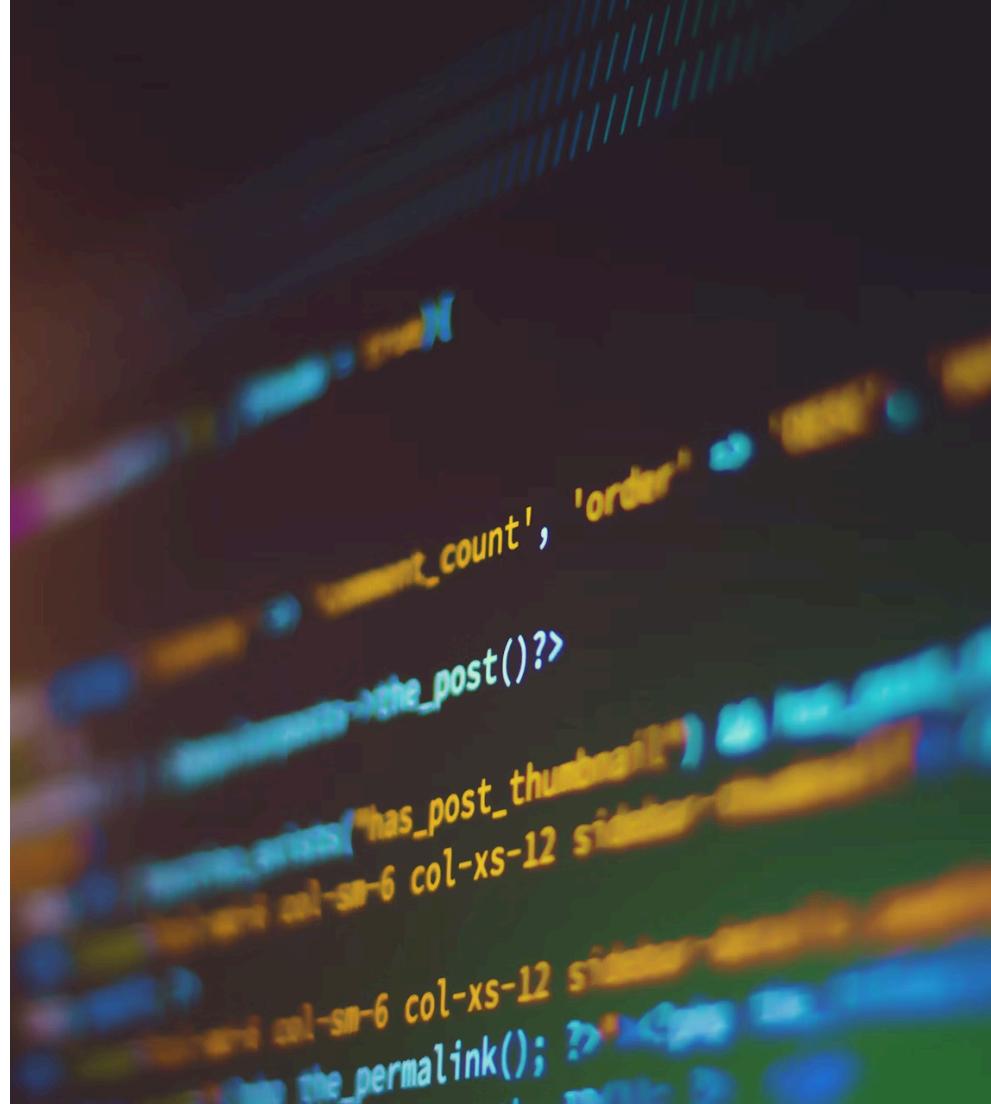
```
# In interactive mode, Codex can search the web
codex
> "Search for the latest React 18 features"
> "Find Python async/await best practices"
> "What are the breaking changes in Spring Boot 3?"
```

- Automatic web search when needed
- Current information beyond training cutoff
- Verify solutions with official docs

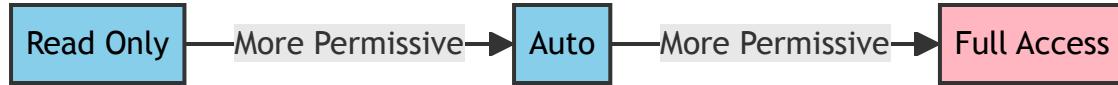
Core Features

Essential Capabilities

Master the fundamentals



Sandbox Modes



- **Read Only** - No file modifications
- **Auto** (default) - Smart restrictions
- **Full Access** - Unrestricted (use carefully!)

Approval Policies

- **on-request** - Approve risky actions
- **never** - No approval prompts
- **on-failure** - Approve only on failures

Setting Safety Options

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

# Set approval policy
codex --approval-policy never

# Bypass all safety (dangerous!)
codex --dangerously-bypass-approvals-and-sandbox
```

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

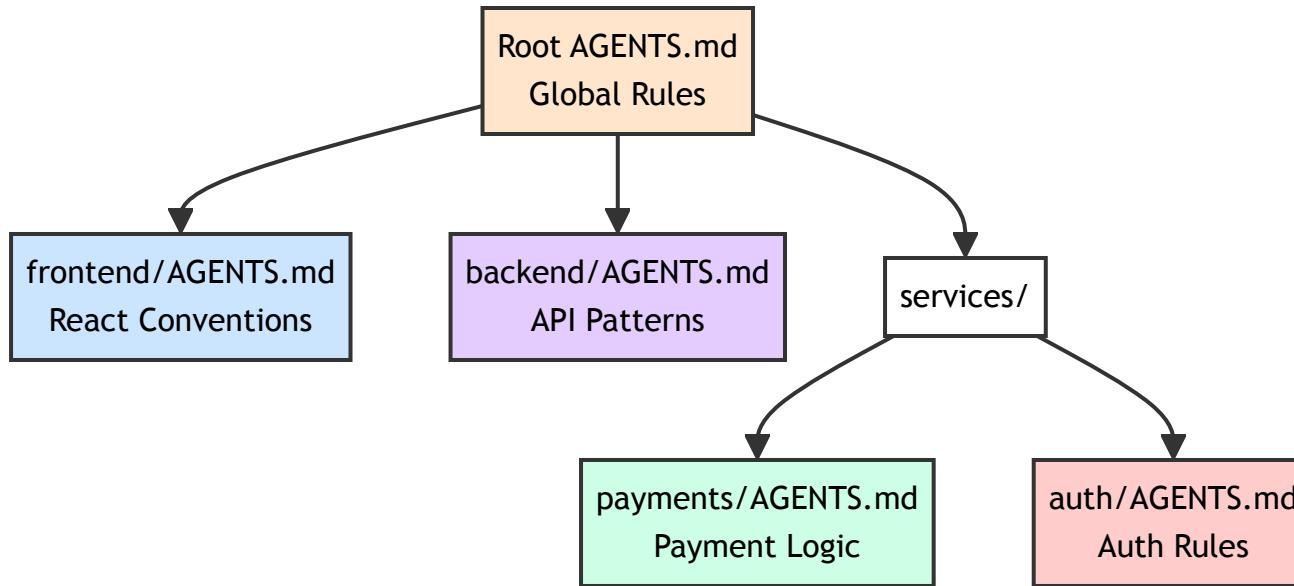
Conventions

- 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 (v0.39+)



Rules cascade: Subfolder overrides parent

Context Cascade Benefits

- Global rules apply everywhere
- Subfolder rules override parent
- 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 Template: Security Audit

```
# ~/.codex/prompts/security-audit.md
```

Perform a comprehensive security audit:

1. Check for SQL injection vulnerabilities
2. Identify hardcoded secrets or API keys
3. Review authentication/authorization logic
4. Scan for XSS vulnerabilities
5. Check for insecure dependencies
6. Review encryption and hashing methods

For each issue found:

- Explain the vulnerability
- Show how to fix it
- Rate severity (Critical/High/Medium/Low)

Prompt Template: Test Generator

```
# ~/.codex/prompts/test-gen.md
```

Generate comprehensive tests for the current code:

1. Unit tests for all public methods
2. Edge cases and error conditions
3. Mock external dependencies
4. Use appropriate assertions
5. Follow AAA pattern (Arrange, Act, Assert)
6. Include performance tests if applicable

Use the project's existing test framework.

Coverage target: 80% minimum

Prompt Template: PR Review

```
# ~/.codex/prompts/pr-review.md
```

Review this code as a senior engineer would:

Check for:

- Bugs and logic errors
- Performance issues
- Security vulnerabilities
- Code style violations
- Missing tests

Provide:

- Line-by-line feedback
- Suggested improvements
- Overall assessment

Prompt Template: API Upgrade

```
# ~/.codex/prompts/api-upgrade.md
```

Migrate this code to use the latest API version:

1. Identify all deprecated method calls
2. Replace with modern equivalents
3. Update import statements
4. Adjust types/interfaces as needed
5. Maintain backward compatibility
6. Add migration notes as comments
7. Update tests to match new API

Show a diff of all changes.

Prompt Template: Performance Fix

```
# ~/.codex/prompts/perf-fix.md
```

Analyze and optimize performance:

1. Identify bottlenecks ($O(n^2)$ loops, etc.)
2. Suggest algorithmic improvements
3. Add caching where appropriate
4. Optimize database queries
5. Reduce memory allocations
6. Consider async/parallel processing

Target: 50% performance improvement

Using Custom Prompts

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

Prompt Arguments Workaround

Codex doesn't support `$ARGUMENTS` like Claude Code, but you can use shell scripts:

```
#!/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`

Configuration Profiles

Define Multiple Profiles

```
# ~/.codex/config.toml

[profiles.production]
model = "gpt-4"
approval_policy = "on-request"
sandbox_mode = "auto"

[profiles.development]
model = "gpt-4o-mini"
approval_policy = "never"
sandbox_mode = "full-access"
```

Using Profiles

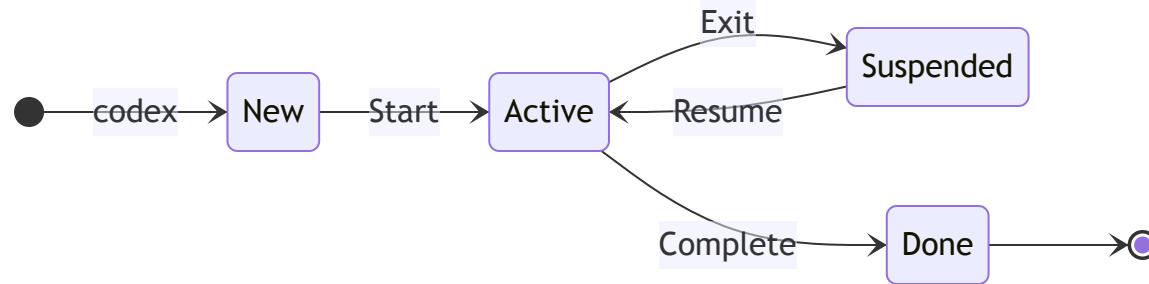
```
codex --profile production  
codex --profile development  
codex --profile testing
```

Resume Previous Sessions

```
# Resume most recent session  
codex --resume
```

```
# Resume specific session  
codex --resume-session abc123
```

Session Lifecycle



Commands: `codex` , `codex --resume` , `codex --list-sessions`

Session Commands

```
# List available sessions
codex --list-sessions
```

```
# Clear old sessions
codex --clear-sessions
```

Non-Interactive Sessions

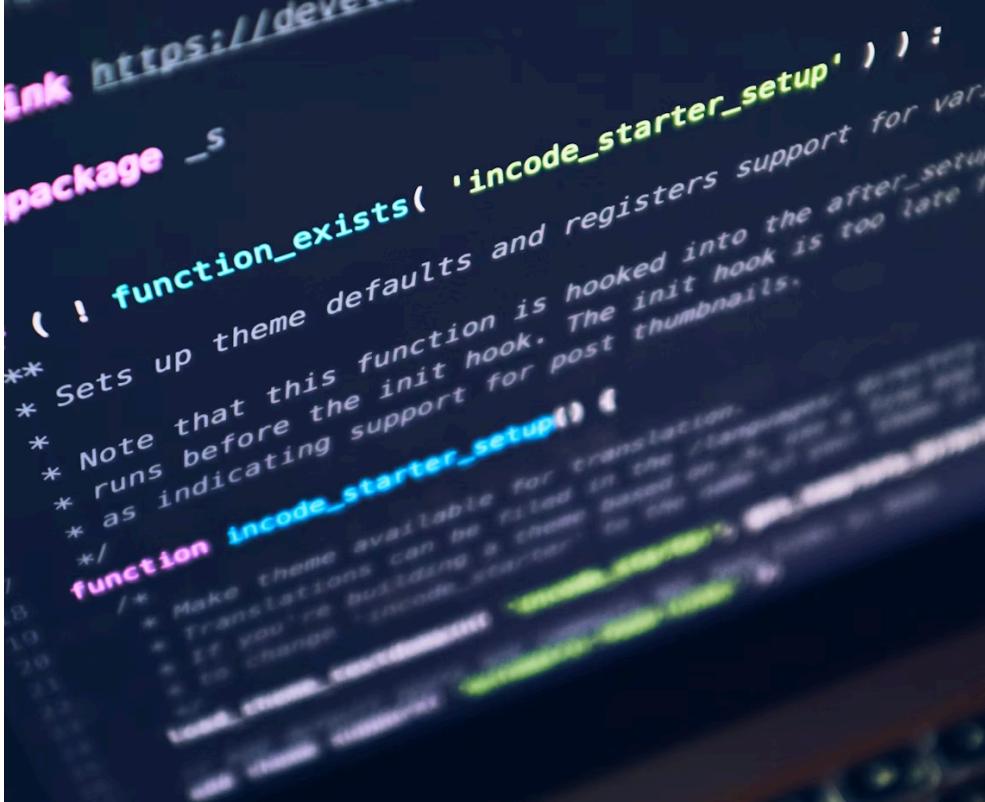
```
# Run in CI/CD pipeline
codex exec "update dependencies and fix breaking changes"

# Note: For resuming, use regular resume
codex --resume
```

Advanced Features

Power User Tools

Unlock full potential



```
link https://devel...
```

```
package _s
```

```
( ! function_exists( 'incode_starter_setup' ) ) {
```

```
* Sets up theme defaults and registers support for var-
```

```
* Note that this function is hooked into the init hook.
```

```
* runs before the init hook. The init hook is after_setu-
```

```
* as indicating support for post thumbnails.
```

```
function incode_starter_setup() {
```

```
/* Make theme available for translation.
```

```
* Translations can be filled in the /languages/
```

```
* file: /languages/your-theme.mo.
```

```
/* If you're building a theme based on inco-
```

```
/* Make sure you have defined the $wp_theme_name
```

```
/* Make sure you have defined the $wp_theme_name
```

Model Context Protocol (MCP)

Configure MCP Servers

GitHub MCP Server

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

Database MCP Server

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

MCP Startup Guardrails (v0.31+)

Prevent flaky tools from freezing Codex:

```
[mcp_servers.github]
command = "npx"
args = ["@modelcontextprotocol/server-github"]
startup_timeout_ms = 15000 # Abort after 15 seconds
```

MCP Timeout Benefits

- Clean abort when helpers fail to boot
- Prevents entire run from freezing
- Better error messages
- Faster feedback on configuration issues

MCP Usage

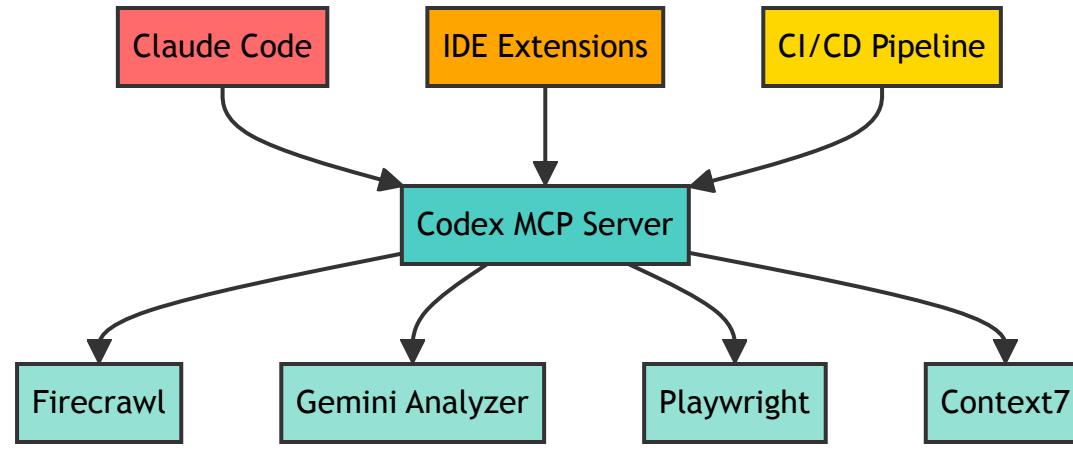
- Automatically available when configured
- Access external tools and data
- Extend Codex capabilities

Running Codex as MCP Server

```
# Modern approach (v0.37+)
codex mcp --config ~/.codex/config.toml

# Legacy approach
codex serve --port 8080
```

MCP Architecture



MCP Server Benefits

The `codex mcp` command exposes Codex as a tool:

- Other agents can call Codex workflows
- IDEs can integrate without plugins
- Claude Code can use as sub-agent
- Mix model strengths (GPT + Claude)

Why Codex as Sub-Agent?

- Leverage GPT-5-Codex for complex tasks
- Use Codex's specialized prompts
- Access different model providers
- Unified approval/sandbox policies

Integration Example

```
# Add Codex as MCP server in Claude Code
claude mcp add codex -- codex mcp

# List MCP servers
claude mcp list

# Remove if needed
claude mcp remove codex
```

Integration Options

- Connect from other MCP clients
- Use in multi-agent workflows
- Integrate with IDEs

Example Client Connection

```
const client = new MCPClient({  
  url: 'http://localhost:8080',  
  capabilities: ['code-generation', 'review']  
});
```

Multi-Model Provider Support

Configure Alternative Providers

Anthropic Claude

```
[providers.anthropic]
type = "anthropic"
api_key = "${ANTHROPIC_API_KEY}"
model = "claude-3-opus-20240229"
```

Local Ollama

```
[providers.ollama]
type = "ollama"
base_url = "http://localhost:11434"
model = "codellama"
```

Azure OpenAI

```
[providers.azure]
type = "azure-openai"
api_key = "${AZURE_API_KEY}"
endpoint = "https://myinstance.openai.azure.com"
deployment = "gpt-4"
```

Switch Providers

```
codex --provider anthropic  
codex --provider ollama  
codex --provider azure
```

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

```
# ~/.codex/config.toml
[logging]
level = "debug"
file = "/path/to/custom.log"
```

CI/CD Integration

GitHub Actions Example

GitHub Actions Setup

```
name: AI Code Review
on: [pull_request]

jobs:
  review:
    runs-on: ubuntu-latest
```

Install and Authenticate

```
steps:  
  - uses: actions/checkout@v3  
  - name: Install Codex  
    run: npm install -g @openai/codex  
  - name: Login to Codex  
    run: codex login --headless  
    env:  
      OPENAI_API_KEY: ${{ secrets.OPENAI_API_KEY }}
```

Run Code Review

```
- name: Review Code
  run: |
    codex exec "Review this PR for bugs,
               performance issues, and best practices"
```

Scheduled Code Maintenance

Use `codex exec` with cron for automated maintenance:

```
# Weekly security check (crontab)
0 2 * * 1 cd /path/to/repo && \
codex exec "identify and fix any security \
vulnerabilities or deprecated API usage"
```

Ambitious: Nightly Dependency Updates

```
# Sunday 3am: Auto-update all dependencies
0 3 * * 0 cd /path/to/repo && \
git pull && \
codex exec "update all dependencies to their \
latest versions and resolve any \
breaking changes" && \
git commit -am "Auto-update dependencies" && \
git push
```

Wake up to a ready-to-review PR! ☀️

Nightly Update Safeguards

- Push to a **separate branch**, not main
- Use **-dry-run** first to test
- Set up **notifications** for failures
- Run **comprehensive tests** before push
- Consider **semantic versioning** limits

```
# Safer version with branch
git checkout -b auto-update-$(date +%Y%m%d) && \
codex exec "update dependencies" && \
npm test && \
git push origin HEAD
```

Maintenance Automation Examples

```
# Daily dependency check
codex exec "check for outdated dependencies \
    and update to latest stable versions"

# Weekly security audit
codex exec "scan for insecure crypto usage, \
    SQL injection risks, or exposed secrets"

# Monthly code cleanup
codex exec "identify dead code, unused imports, \
    and refactor for better performance"
```

Automated Maintenance Best Practices

- Always review changes before merging
- Use sandbox mode for safety
- Create separate branch for fixes
- Set up notifications for runs
- Keep audit logs of changes

Remember: Codex drafts solutions, you review and approve

Chain Commands with &&

Create fail-fast pipelines using && :

```
# Only proceed if each step succeeds
git pull && \
codex exec "migrate database schema to latest version" && \
npm test
```

If any step fails, execution stops immediately

Command Chaining Examples

```
# Update, migrate, test
git pull && \
codex exec "update dependencies to fix vulnerabilities" && \
codex exec "migrate breaking changes" && \
npm test && \
git commit -am "Update dependencies"

# Build pipeline
npm install && \
codex exec "fix any TypeScript errors" && \
npm run build && \
npm test
```

Why Use && Chaining?

- **Fail-fast behavior** - Stop on first error
- **Dependency ordering** - Each step needs the previous
- **Atomic operations** - All succeed or nothing commits
- **CI/CD friendly** - Clear success/failure signal
- **Prevents partial updates** - No broken states

Alternative: Use `||` for fallback commands

Advanced TOML Configuration

```
# ~/.codex/config.toml
model = "gpt-4o"
model_provider = "openai"
approval_policy = "on-request"
sandbox_mode = "auto"

[tools]
web_search = true
```

Environment & Notifications

```
[notification]
program = "notify-send"
args = ["Codex", "Task completed"]

[shell_environment]
NODE_ENV = "development"
PYTHONPATH = "/usr/local/lib/python3.9"
```

Shell Environment Policies

- **inherit** - Use parent shell environment
- **explicit** - Only specified variables
- **minimal** - Basic environment only

Environment Configuration

```
# Inherit all variables
shell_environment_policy = "inherit"

# Explicit variables only
shell_environment_policy = "explicit"
```

Explicit Environment

```
[shell_environment]
PATH = "/usr/local/bin:/usr/bin:/bin"
HOME = "/home/user"
LANG = "en_US.UTF-8"
```

Security Considerations

- Use `explicit` for production
- Use `inherit` for development
- Never expose secrets in config

Notification System

Configure External Notifications

macOS Notifications

```
[notification]
program = "osascript"
args = ["-e", "display notification \"Task complete\" with title \"Codex\""]
```

Linux Notifications

```
[notification]
program = "notify-send"
args = ["Codex", "Task completed"]
```

Webhook Notifications

```
[notification]
program = "curl"
args = ["-X", "POST", "https://hooks.slack.com/...", "-d", "{\"text\":\"Done\"}"]
```

Notification Triggers

- On task completion
- On approval required
- On errors (configurable)



Practical Exercises

Hands-On Labs

Learn by doing

Exercise Structure

Three Main Categories

1. Generate from Scratch

- Spring Boot REST API
- Python CLI tool
- React dashboard

2. Refactor Existing Code

- Legacy Java modernization
- Python code quality improvements
- TypeScript migration

3. Multi-Language Projects

- Microservices architecture
- Full-stack application
- Data pipeline

Each Exercise Includes

- Starting code
- Requirements
- Expected outcomes
- Solution walkthrough

Lab 1: Spring Boot API

Build a Complete REST API

Spring Boot Requirements

1. REST endpoints for CRUD operations
2. H2 in-memory database
3. Input validation
4. Exception handling
5. OpenAPI documentation
6. Comprehensive tests

Starting Prompts

```
# Start with project structure
codex "Generate a Spring Boot 3.2 project
      structure for a task management API"
```

Build the API

```
# Add domain model
codex "Create JPA entities for Task"

# Generate complete API
codex "Implement REST controllers with
proper error handling and validation"
```

Lab 2: Python Refactoring

Transform Legacy Code

Starting Legacy Code

```
# calculator.py - needs refactoring
def calc(x,y,op):
    if op=="add": return x+y
    elif op=="sub": return x-y
    elif op=="mul": return x*y
    elif op=="div": return x/y
```

Refactoring Goals

- Add type hints
- Proper error handling
- Strategy pattern
- Comprehensive docstrings
- pytest test suite with 100% coverage

Codex Refactoring

```
codex "Refactor this calculator to use  
modern Python best practices"
```

Lab 3: React TypeScript Forms

Build a Registration System

Form Requirements

- Email/password validation using Zod
- React Hook Form integration
- Accessibility (ARIA) compliance
- Jest + React Testing Library tests
- Responsive Tailwind CSS styling

Progressive Enhancement

1. Start with basic form
2. Add validation
3. Implement testing
4. Add accessibility
5. Optimize performance

Generate the Form

```
codex "Create a React TypeScript  
registration form with Zod validation  
and full test coverage"
```

Lab 4: Microservices

Multi-Service Architecture

Configure Project Context

```
echo "# Microservices Project  
Tech: Node.js, Python, Go  
Pattern: Event-driven with RabbitMQ  
Deploy: Docker Compose" > AGENTS.md
```

Service Components

- Auth Service (Node.js) - JWT authentication
- Order Service (Python FastAPI) - Order processing
- Notification Service (Go) - Email/SMS alerts

Generate Services

```
codex "Create microservices architecture with  
Docker setup and message queue integration"
```

Key Learning Points

- Cross-language generation
- Service communication
- Container orchestration

Lab 5: Database Migration

Modernize Legacy Database

Setup Database MCP

```
codex "Configure MCP server for PostgreSQL connection"
```

Migration Steps

```
# Analyze schema
codex "Analyze database schema"

# Create migration plan
codex "Create migration to normalized structure"

# Generate scripts
codex "Generate Flyway migration scripts"
```

Skills Practiced

- MCP integration
- Database analysis
- Safe migrations

Lab 6: AI Code Review

Automated PR Reviews

Workflow Configuration

```
# .github/workflows/ai-review.yml
name: Codex Review
on: [pull_request]

jobs:
  ai-review:
    runs-on: ubuntu-latest
```

Setup Review Profile

```
- name: Configure Codex
  run: |
    echo "[profiles.review]"
    sandbox_mode = \"read-only\"
    approval_policy = \"never\" > ~/.codex/config.toml
```

Run Review

```
- name: Review Changes
  run: |
    codex exec --profile review \
    "Review for: security issues,
     performance problems, best practices"
```

Advanced Exercise

Full-Stack E-Commerce

Phase 1: Backend

```
codex "Create Spring Boot backend with:  
- Product catalog management  
- User authentication (JWT)  
- Order processing workflow  
- PostgreSQL with Liquibase"
```

Phase 2: Frontend

```
codex "Create React frontend with:  
- Product browsing with filters  
- Shopping cart (Redux)  
- Checkout process  
- Admin dashboard"
```

Phase 3: DevOps

```
codex "Add Docker Compose setup with  
nginx, monitoring, and CI/CD pipeline"
```

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

```
[profiles.dev]
sandbox_mode = "full-access"
approval_policy = "never"
```

Staging Profile

```
[profiles.staging]
sandbox_mode = "auto"
approval_policy = "on-request"
```

Production Profile

```
[profiles.prod]
sandbox_mode = "read-only"
approval_policy = "on-request"
```

Security Guidelines

- Never store API keys in config files
- Use environment variables for secrets
- Enable approval for production
- 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

```
[profiles.quick]
model = "gpt-3.5-turbo" # Fast responses
```

```
[profiles.complex]
model = "gpt-4" # Complex reasoning
```

Local Models

```
[profiles.local]
provider = "ollama"
model = "codellama" # No API costs
```

Optimization Tips

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

Troubleshooting Guide

Common Issues & Solutions

Authentication Failures

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

```
# Re-authenticate
codex login --headless
```

Sandbox Errors

```
# Check sandbox support
codex doctor

# Bypass for Docker environments
codex --dangerously-bypass-approvals-and-sandbox
```

MCP Connection Issues

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

# Enable debug logging
RUST_LOG=trace codex
```

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 -n --profile review \
    "Check staged files for security issues"
```

Make Integration

```
review:
```

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

```
generate-tests:
```

```
codex exec "Generate missing unit tests"
```

Structured Logging

```
[logging]
level = "info"
format = "json"
file = "/var/log/codex/codex.log"
```

Metrics Configuration

```
[metrics]
enable = true
endpoint = "http://metrics.internal:9090"
```

Log Analysis

```
# Parse JSON logs
cat ~/.codex/log/codex-tui.log | jq '.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

```
[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 Claude Code

```
# Import Claude memory
cp ./claude-code/.claude/CLAUDE.md ./AGENTS.md

# Convert slash commands
for f in ./claude-code/.claude/commands/*.md; do
    cp "$f" ~/.codex/prompts/
done
```

From Cursor/Codeium

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

Recent Features (v0.30-0.39)

Version 0.30-0.39 Highlights

- MCP startup timeouts (v0.31)
- Token usage in /status command (v0.35)
- GPT-5-Codex high reasoning mode (v0.36)
- Network allowlists for testing (v0.36)
- Simplified MCP server: `codex mcp` (v0.37)
- Hierarchical AGENTS.md cascading (v0.39)
- Enhanced error handling
- Improved session management
- Performance optimizations

MCP Robustness (v0.31+)

Always add timeout to MCP servers:

```
[mcp_servers.your_server]
command = "your-command"
startup_timeout_ms = 15000 # Recommended
```

Advanced Model Control (v0.36+)

Use GPT-5-Codex for complex, long-running tasks:

```
codex -m gpt-5-codex -c model_reasoning_effort='high'
```

Or configure in TOML:

```
model_reasoning_effort = "high" # minimal/low/medium/high
```

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
```

Note: Domain-specific allowlists may be available in v0.36+ (check release notes)

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

Q1 2025 Roadmap

- Plugin ecosystem
- Visual Studio Code extension
- Enhanced MCP marketplace

Q2 2025 Roadmap

- Multi-agent coordination
- Real-time collaboration
- Advanced code analysis

Future Vision

- Voice control integration
- AR/VR code visualization
- Quantum computing support

Community Contributions

- Open source at github.com/openai/codex
- Feature requests welcome
- Plugin development SDK coming

Essential Commands

```
# Basic usage
codex                         # Interactive mode
codex exec "prompt"            # Execute task & exit
codex --resume                  # Resume session
codex --search "text"           # Search codebase
```

Configuration Commands

```
codex --profile production    # Use profile  
codex --sandbox-mode auto      # Set sandbox  
codex --approval-policy never # Set approval
```

Advanced Commands

```
codex mcp          # MCP server mode (v0.37+)
codex doctor       # Diagnostics
codex --list-sessions # Show sessions
```

Key Files

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

Documentation & Code

Official Documentation

<https://github.com/openai/codex/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>

Discord & Office Hours

Discord Server

Join the Codex community for support and discussion

Weekly Office Hours

Every Thursday at 2 PM PST

Contributing

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

Ecosystem

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

References & Credits

Newsletter Sources

MLearning.ai Art on Substack by @mlearning

- [100 OpenAI Codex CLI Tricks and Tips](#)
- [30 Codex CLI Tips v0.30-0.39](#)

Many advanced tips including:

- Command chaining with `&&`
- Scheduled maintenance automation
- Network allowlists
- High reasoning mode

Additional Resources

Official Sources

- [Codex GitHub Repository](#)
- [Codex Documentation](#)
- [Model Context Protocol](#)

Community

- [Codex Discussions](#)
- [MCP Registry](#)

Related Training

- [Claude Code Training](#)
- [Junie Training Materials](#)

Thank You!

Questions?



Kenneth Kousen*Author, Speaker, Java & AI Expert*

kousenit.com | [@kenkousen](https://twitter.com/kenkousen)