



Basic Configuration

Codex reads local settings from `~/ .codex/config.toml`. Use this file to change defaults (like the model), set approval and sandbox behavior, and configure MCP servers.

Codex configuration file

Codex stores its configuration at `~/ .codex/config.toml`.

To open the configuration file from the Codex IDE extension, select the gear icon in the top-right corner, then select **Codex Settings > Open config.toml**.

The CLI and IDE extension share the same `config.toml` file. You can use it to:

- Set the default model and provider.
- Configure [approval policies and sandbox settings](#).
- Configure [MCP servers](#).

Configuration precedence

Codex resolves values in this order:

1. CLI flags (for example, `--model`)
2. [Profile](#) values (from `--profile <name>`)
3. Root-level values in `config.toml`
4. Built-in defaults

Use that precedence to set shared defaults at the top level and keep profiles focused on the values that differ.

For one-off overrides via `-c/--config` (including TOML quoting rules), see [Advanced Config](#).

On managed machines, your organization may also enforce constraints via `requirements.toml` (for example, disallowing `approval_policy = "never"` or `sandbox_mode = "danger-full-access"`). See [Security](#).

Common configuration options

Here are a few options people change most often:

Default model

Choose the model Codex uses by default in the CLI and IDE.

```
model = "gpt-5.2"
```

Approval prompts

Control when Codex pauses to ask before running generated commands.

```
approval_policy = "on-request"
```

Sandbox level

Adjust how much filesystem and network access Codex has while executing commands.

```
sandbox_mode = "workspace-write"
```

Reasoning effort

Tune how much reasoning effort the model applies when supported.

```
model_reasoning_effort = "high"
```

Command environment

Restrict or expand which environment variables are forwarded to spawned commands.

```
[shell_environment_policy]  
include_only = ["PATH", "HOME"]
```

Feature flags

Optional and experimental capabilities are toggled via the [features] table in config.toml.

```
[features]  
shell_snapshot = true           # Speed up repeated commands
```

```
web_search_request = true      # Allow the model to request web
                               searches
```

Supported features

Key	Default	Maturity	Description
apply_patch_freeform	false	Experimental	Include the freeform apply_patch tool
elevated_windows_sandbox	false	Experimental	Use the elevated Windows sandbox pipeline
exec_policy	true	Experimental	Enforce rules checks for shell/unified_exec
experimental_windows_sandbox	false	Experimental	Use the Windows restricted-token sandbox
remote_compaction	true	Experimental	Enable remote compaction (ChatGPT auth only)
remote_models	false	Experimental	Refresh remote model list before showing readiness
shell_snapshot	false	Beta	Snapshot your shell environment to speed up repeated commands
shell_tool	true	Stable	Enable the default shell tool
unified_exec	false	Beta	Use the unified PTY-backed exec tool
undo	true	Stable	Enable undo via per-turn git ghost snapshots
web_search_request	false	Stable	Allow the model to issue web searches

The Maturity column uses feature maturity labels such as Experimental, Beta, and Stable. See [Feature Maturity](#) for how to interpret these labels.

Omit feature keys to keep their defaults.

Enabling features quickly

- In `config.toml`, add `feature_name = true` under `[features]`.
- From the CLI, run `codex --enable feature_name`.
- To enable multiple features, run `codex --enable feature_a --enable feature_b`.
- To disable a feature, set the key to `false` in `config.toml`.