Codex

Copy page

Delegate tasks to a software engineering agent in the cloud.

Codex is a cloud-based software engineering agent. Use it to fix bugs, review code, do refactors, and fix pieces of code in response to user feedback. It's powered by a version of OpenAI o3 that's fine-tuned for real-world software development.

Overview

We believe in a future where developers drive the work they want to own, delegating toilsome tasks to agents. We see early signs of this future today at OpenAI, with Codex working in its own environment and drafting pull requests in our repos.

Codex vs. Codex CLI
These docs cover Codex, a cloud-based agent you can find in your browser. For an open-source CLI agent you can run locally in your terminal, install Codex CLI.

Video: Getting started with Codex

Codex evolves quickly and may not match exactly the UI shown below, but this video will give you a quick overview of how to get started with Codex inside ChatGPT.

0:00 / 4:25

Connect your GitHub

To grant the Codex agent access to your GitHub repos, install our GitHub app to your organization. The two permissions required are ability to *clone the repo* and the ability to *push a pull request* to it. Our app **will not write to your repo without your permission.**

Each user in your organization must authenticate with their GitHub account before being able to use Codex. After auth, we grant access to your GitHub repos and environments at the ChatGPT workspace level—meaning if your teammate grants access to a repo, you'll also be able to run Codex tasks in that repo, as long as you share a workspace.

How it works

At a high level, you specify a prompt, and the agent goes to work in its own environment. After about 8-10 minutes, the agent gives you back a diff.

You can execute prompts in either *ask* mode or *code* mode. When you select *ask*, Codex clones a read-only version of your repo, booting faster and giving you follow-up tasks. *Code* mode, however, creates a full-fledged environment that the agent can run and test against.

Exact flow:

1. You navigate to chatgpt.com/codex and **submit a task.**
2. We launch a new **Docker container** based upon our **base image**. We then **clone your repo** at the desired **branch or sha** and run any **setup scripts** you have from the specified **workdir**.
3. We **disable network access**. The agent does not have the ability to install dependencies from this point forward.
4. The agent then **runs terminal commands in a loop**. It writes code, runs tests, and attempts to check its work. The agent attempts to honor any specified lint or test commands you've defined in an `AGENTS.md` file. The agent does not have access to any special tools outside of the terminal or CLI tools you provide.
5. When the agent completes your task, it **presents a a diff** or a set of follow-up tasks. You can choose to **open a PR** or respond with follow-up comments to ask for additional changes.

Submit tasks to Codex

After connecting your repository, begin sending tasks using one of two modes:

Ask mode for brainstorming, audits, or architecture questions

Code mode for when you want automated refactors, tests, or fixes applied

Below are some example tasks to get you started with Codex.

Ask mode examples

Use ask mode to get advice and insights on your code, no changes applied.

1 **Refactoring suggestions**

Codex can help brainstorm structural improvements, such as splitting files, extracting functions, and tightening documentation.

```
Take a look at <hairiest file in my codebase>.
Can you suggest better ways to split it up, test it, and isolate functionali
```

2 **Q&A and architecture understanding**

Codex can answer deep questions about your codebase and generate diagrams.

```
Document and create a mermaidjs diagram of the full request flow from the cl
endpoint to the database.
```

Code mode examples

Use code mode when you want Codex to actively modify code and prepare a pull request.

1 **Security vulnerabilities**

Codex excels at auditing intricate logic and uncovering security flaws.

```
There's a memory-safety vulnerability in <my package>. Find it and fix it
```

2 **Code review**

Append `.diff` to any pull request URL and include it in your prompt. Codex loads the patch inside the container.

```
Please review my code and suggest improvements. The diff is below:
<diff>
```

3 **Adding tests**

After implementing initial changes, follow up with targeted test generation.

```
From my branch, please add tests for the following files:
<files>
```

4 **Bug fixing**

A stack trace is usually enough for Codex to locate and correct the problem.

```
Find and fix a bug in <my package>.
```

5 **Product and UI fixes**

Although Codex cannot render a browser, it can resolve minor UI regressions.

```
The modal on our onboarding page isn't centered. Can you fix it?
```

Environment configuration

While Codex works out of the box, you can customize the agent's environment to e.g. install dependencies and tools. Having access to a fuller set of dependencies, linters, formatters, etc. often results in better agent performance.

Default universal image

The Codex agent runs in a default container image called `universal`, which comes pre-installed with common languages, packages, and tools.

Set package versions in environment settings can be used to configure the version of Python, Node.js, etc.

openai/codex-universal
For details on what's installed, see `openai/codex-universal` for a reference Dockerfile and an image that can be pulled and tested locally.

While `codex-universal` comes with languages pre-installed for speed and convenience, you can also install additional packages to the container using setup scripts.

Environment variables and secrets

Environment variables can be specified and are set for the full duration of the task.

Secrets can also be specified and are similar to environment variables, except:

- They are stored with an additional layer of encryption and are only decrypted for task execution.
- They are only available to setup scripts. For security reasons, secrets are removed from the environment when the agent is running.

Setup scripts

Setup scripts are bash scripts that run at the start of every task. To get the best results from the agent, we recommend installing dependencies and linters / code formatters—and using AGENTS.md to instruct the agent to run tests and use these tools.

```
# Install type checker
pip install pyright
# Install dependencies
poetry install --with test
pnpm install
```

Setup scripts are run in a separate bash session than the agent, so commands like `export` do not persist. You can persist environment variables by adding them to `~/.bashrc`.

Internet access and network proxy

Internet access is currently only available during the setup script phase, so dependencies should be installed up front in setup scripts.

When internet access *is* available, environments run behind an HTTP/HTTPS network proxy for security and abuse prevention purposes. All outbound internet traffic must pass through this proxy.

Environments are pre-configured to work with common tools and package managers:

1. Codex sets standard environment variables including `http_proxy` and `https_proxy`. These settings are respected by tools such as `curl`, `npm`, and `pip`.
2. Codex installs a proxy certificate into the system trust store. This certificate's path is available as the environment variable `$CODEX_PROXY_CERT`. Additionally, specific package manager variables (e.g., `PIP_CERT`, `NODE_EXTRA_CA_CERTS`) are set to this certificate path.

If you're encountering connectivity issues, verify and/or configure the following:

- Ensure you are connecting via the proxy at `http://proxy:8080`.
- Ensure you are trusting the proxy certificate located at `$CODEX_PROXY_CERT`. Always reference this environment variable instead of using a hardcoded file path, as the path may change.

Using AGENTS.md

Provide common context by adding an `AGENTS.md` file. This is just a standard Markdown file the agent reads to understand how to work in your repository. `AGENTS.md` can be nested, and the agent will by default respect whatever the most nested root that it's looking for. Some customers also prompt the agent to look for `.currsorrules` or `CLAUDE.md` explicitly. We recommend sharing any bits of organization-wide configuration in this file.

Common things you might want to include:

- An overview showing which particular files and folders to work in

Contribution and style guidelines

Parts of the codebase being migrated

How to validate changes (running lint, tests, etc.)

How the agent should do and present its work (where to explore relevant context, when to write docs, how to format PR messages, etc.)

Here's an example as one way to structure your `AGENTS.md` file:

AGENTS.md

```
# Contributor Guide

## Dev Environment Tips
- Use pnpm dlx turbo run where <project_name> to jump to a package instead of scann
- Run pnpm install --filter <project_name> to add the package to your workspace so
- Use pnpm create vite@latest <project_name> -- --template react-ts to spin up a ne
- Check the name field inside each package's package.json to confirm the right name

## Testing Instructions
- Find the CI plan in the .github/workflows folder.
- Run pnpm turbo run test --filter <project_name> to run every check defined for th
- From the package root you can just call pnpm test. The commit should pass all tes
- To focus on one step, add the Vitest pattern: pnpm vitest run -t "<test name>".
- Fix any test or type errors until the whole suite is green.
- After moving files or changing imports, run pnpm lint --filter <project_name> to
- Add or update tests for the code you change, even if nobody asked.

## PR instructions
Title format: [<project_name>] <Title>
```

Prompting Codex

Just like ChatGPT, Codex is only as effective as the instructions you give it. Here are some tips we find helpful when prompting Codex:

Provide clear code pointers

Codex is good at locating relevant code, but it's more efficient when the prompt narrows its search to a few files or packages. Whenever possible, use **greppable identifiers, full stack traces, or rich code snippets**.

Include verification steps

Codex produces higher-quality outputs when it can verify its work. Provide **steps to reproduce an issue, validate a feature, and run any linter or pre-commit checks**. If additional packages or custom setups are needed, see Environment configuration.

Customize how Codex does its work

You can **tell Codex how to approach tasks or use its tools**. For example, ask it to use specific commits for reference, log failing commands, avoid certain executables, follow a template for PR messages, treat specific files as AGENTS.md, or draw ASCII art before finishing the work.

Split large tasks

Like a human engineer, Codex handles really complex work better when it's broken into smaller, focused steps. Smaller tasks are easier for Codex to test and for you to review. You can even ask Codex to help break tasks down.

Leverage Codex for debugging

When you hit bugs or unexpected behaviors, try **pasting detailed logs or error traces into Codex as the first debugging step**. Codex can analyze issues in parallel and could help you identify root causes more quickly.

Try open-ended prompts

Beyond targeted tasks, Codex often pleasantly surprises us with open-ended tasks. Try asking it to clean up code, find bugs, brainstorm ideas, break down tasks, write a detailed doc, etc.