CodeArts Agent

User Guide (CLI)

Issue	01
Date	2026-08-03

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 What Is CodeArts Agent CLI?....1
2 Installation and Upgrade....3
2.1 Prerequisites....3
2.2 Installation....5
2.3 Authorization....6
2.4 Upgrade....10
3 Permissions....11
4 Custom Commands....14
5 Sandbox....18
6 Agents....29
7 Skills....36
8 Hooks....39
9 Repository Indexing....47
10 LSP....53
11 Scheduled Tasks....55
12 Custom Models....65
13 MCP....72
13.1 Default MCP Server Configuration....72
13.2 Claude Code....76
14 TUI....80
14.1 Slash Commands....80
14.2 Shortcut Keys....83
14.3 SDD....83
15 CLI....89
15.1 Commands....89
15.2 MCP Commands....102

1 What Is CodeArts Agent CLI?

Overview

CodeArts Agent CLI is an intelligent coding assistant for developers. It provides end-to-end (E2E) intelligent capabilities, such as code writing, analysis, and optimization.

Both terminal user interface (TUI) and command-line interface (CLI) are available to seamlessly integrate into various R&D workflows and adapt to diverse development scenarios. In headless environments, TUI allows you to conveniently write and debug code in a visualized manner, while CLI allows you to quickly compile code using commands.

Table 1-1 Differences between TUI and CLI

Item	TUI	CLI
Interaction	Dialog box input, visual operations	Direct parameter input, one-line command execution
Scenarios	Interactive development, code writing, and debugging	Script automation and batch tasks
Typical users	New users and routine development	Developers with command line experience

Supported OSs

- Windows: Windows 11 (x64) is recommended. For Windows 10 (x64), the version must be 2019 or later, and upgrading to the latest stable version is recommended.
- macOS: macOS 11 or later, compatible with ARM64 (Apple Silicon)
- Linux: Huawei Cloud EulerOS 2.0, SUSE Linux Enterprise Server 12 SP5, Ubuntu 18.04/20.04/22.04/24.04 LTS, Debian 10/11/12, CentOS 8, RHEL 8/9

Core Functions

- Command coverage: Core commands such as models (model configuration) and sessions (session management) are provided to support full lifecycle management.
- Script-based integration: Tasks can be directly transferred through command parameters. Alternatively, commands can be written into script files (such as shell or Python scripts) for automatic execution and batch processing of AI tasks.

Advantages

- Dual interaction modes: Both TUI and CLI are provided to support interactive visual operations and automatic script calls, adapting to different user preferences and service scenarios.
- Intelligent and scalable agent framework: Built-in core AI agents (such as Build and Plan) and custom agent orchestration flexibly adapt to complex project development and task decomposition needs.
- Rich ecosystem integration: Native support for extended capabilities such as Model Context Protocol (MCP) servers, skills, and custom agents enables deep integration with various development tools and service capabilities.
- Flexible and standardized configuration: Configuration files in JSON and JSONC formats are supported with a clear, readable structure. This streamlines local debugging, team synchronization, and repository hosting.

2 Installation and Upgrade

2.1 Prerequisites

Supported OSs

For optimal compatibility and performance, the following OSs are recommended:

- Windows: **Windows 11 (x64) is recommended.** For Windows 10 (x64), the version must be 2019 or later, and upgrading to the latest stable version is recommended.
- macOS: macOS 11 or later, compatible with ARM64 (Apple Silicon).
- Linux: Huawei Cloud EulerOS 2.0, SUSE Linux Enterprise Server 12 SP5, Ubuntu 18.04/20.04/22.04/24.04 LTS, Debian 10/11/12, CentOS 8, RHEL 8/9

Purchasing a Package

Step 1 Sign up for Huawei Cloud.

1. Visit the **Huawei Cloud official website** and click **Sign Up** in the upper right corner.
2. **Sign up for a HUAWEI ID and enable Huawei Cloud services**.

Step 2 **To purchase a Basic or Professional edition package, complete real-name authentication first.**

Go to the **CodeArts Agent purchase page** and subscribe to a package.

- The account that subscribes to the package has enterprise administrator permissions and can be used to install CodeArts Agent CLI by referring to **Installation**.
- If you use a HUAWEI ID and want IAM users to use CodeArts Agent CLI, **create IAM users using your HUAWEI ID**. Then, the **enterprise administrator** can **add them as enterprise members and enable their seats**.

----End

Adding Enterprise Members and Enabling Seats

Step 1 Log in to the **CodeArts Agent** console using **HUAWEI ID**.

Step 2 In the navigation pane on the left, choose **Enterprise Management** > **Teams & Members**.

Step 3 Click **Add Member** in the upper right corner. The **Add Member** dialog box is displayed.

Step 4 Configure enterprise member information by referring to **Table 2-1**.

Available seats will be automatically assigned to new members. Members added in batches are assigned seats in alphabetical order. If seats run out, priority is given from the top of the list. You can reallocate seats on the **Seats** page for your members or change the package to obtain more seats. Only members with seats can use the CodeArts Agent services.

Figure 2-1 Adding an existing IAM user as an enterprise member

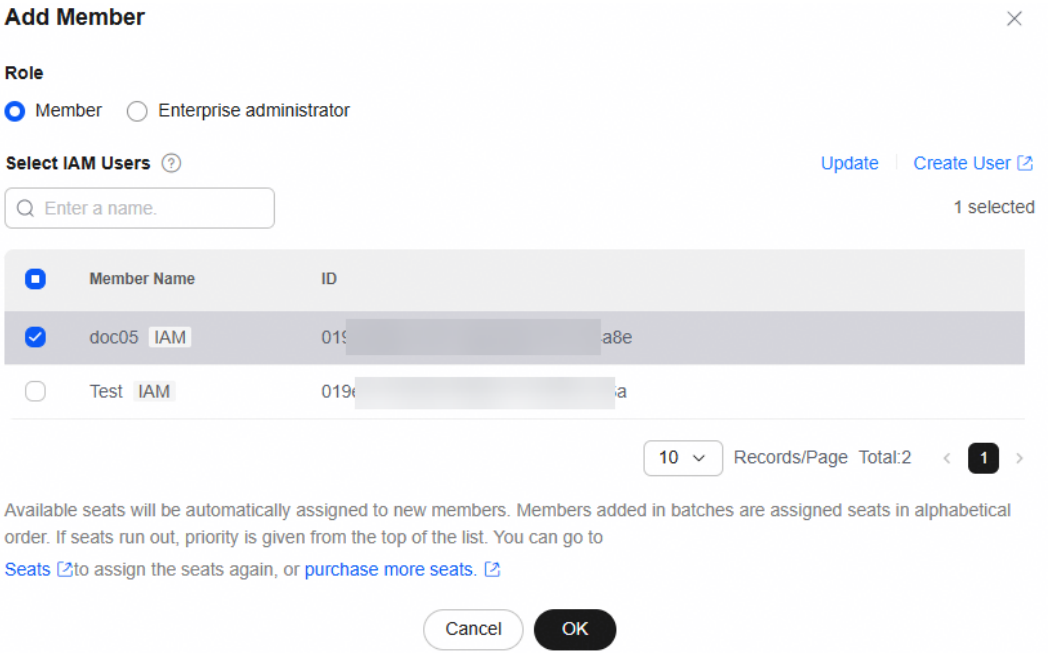


Table 2-1 Parameters for adding an existing IAM user as an enterprise member

Parameter	Description
Role	Indicates the role of the IAM user in CodeArts Agent. • **Member**: Has common member permissions and can perform operations such as viewing the usage of their own and viewing knowledge spaces. • **Enterprise administrator**: Has the highest management permissions and can perform operations such as adding enterprise members, creating teams, and assigning seats.

Parameter	Description
Select IAM Users	Select the IAM user to be added from the IAM user list. You can select multiple IAM users at a time.

Step 5 Click **OK**.

On the **All Members** list, view the new member.

----End

2.2 Installation

This section guides you through a first-time download and installation, using Linux screenshots for demonstration.

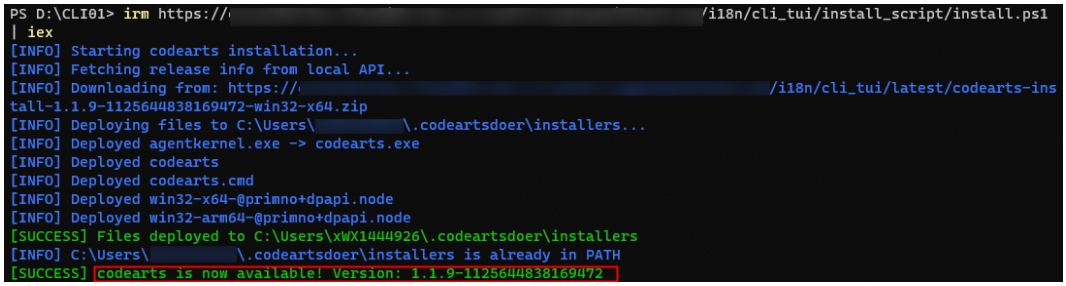
Windows

Step 1 Run the following command in PowerShell (**Windows** only supports installation command execution in **PowerShell**):

```
irm https://apsoutheast1-cloudide-marketplace.obs.ap-southeast-1.myhuaweicloud.com/codearts/cli_tui/install_script/install.ps1 | iex
```

If the CodeArts Agent CLI version is displayed, as shown in the following figure, CodeArts Agent CLI has been successfully installed.

Figure 2-2 Output indicating successful installation on Windows



Step 2 After the installation is successful, complete authorization by referring to **Authorization**.

Step 3 (Optional) If you are on the intranet, **configure the proxy**.

----End

macOS/Linux

The following uses Linux screenshots as examples for demonstration.

Step 1 (Optional) If you are on the intranet, **configure the proxy**.

Step 2 Run the following command on the terminal to download and execute the installation script of CodeArts Agent CLI:

```
sh -c "$(curl -L https://apsoutheast1-cloudide-marketplace.obs.ap-southeast-1.myhuaweicloud.com/codearts/cli_tui/install_script/install.sh)" && export PATH=~/.codeartsdoer/installers:$PATH
```


If the CodeArts Agent CLI version is displayed, as shown in the following figure, CodeArts Agent CLI has been successfully installed.

Figure 2-3 Output indicating successful installation

Step 3 After the installation is successful, complete authorization by referring to **Authorization**.

----End

2.3 Authorization

In CodeArts Agent CLI, you can perform authorization by configuring **process-level variables** or **system environment variables**. The process-level configuration is used for temporary authorization, and the system environment variable configuration is used for permanent authorization.

Preparations

Step 1 Create access keys by referring to **Access Keys**.

Step 2 In the **Create Access Key** dialog box, select the agreement check box and click **Next**. Enter a description for the access key and click **OK**.

After the access key is created, download it and keep it secure. Once the dialog box is closed, the key cannot be obtained.

Step 3 Configure environment variables based on your operating system (see the following). Obtain the values of Access Key Id and Secret Access Key from the downloaded credentials.csv file.

----End

Windows

The following describes how to perform authorization by configuring process-level variables and system environment variables in Windows.

Configuring Process-Level Variables

You can configure process-level variables to grant temporary authorization for the current window.

- If you are using CMD, run the following commands to configure variables:

  ```
  set CODEARTS_CLI_AK=Access Key Id
  set CODEARTS_CLI_SK=Secret Access Key
  ```

  After the commands are executed, run the following commands in the current window to check whether the configuration has taken effect:


```
echo %CODEARTS_CLI_AK%
echo %CODEARTS_CLI_SK%
```

If the content you just set is displayed, the configuration has taken effect. If "%CODEARTS_CLI_AK%" or "%CODEARTS_CLI_SK%" is displayed, the configuration has not taken effect.

- If you are using PowerShell, run the following commands to configure variables:

```
$env:CODEARTS_CLI_AK="Access Key Id"
$env:CODEARTS_CLI_SK="Secret Access Key"
```

After the commands are executed, run the following commands to check whether the configuration has taken effect. If no error message is displayed, the configuration has taken effect.

```
echo $env:CODEARTS_CLI_AK
echo $env:CODEARTS_CLI_SK
```

Configuring System Environment Variables

You can configure system environment variables to make the configuration permanently effective.

Step 1 Open the **System Properties** dialog box and click **Environment Variables** on the **Advanced** tab.

Step 2 Create **CODEARTS_CLI_AK** and **CODEARTS_CLI_SK** under user variables or system variables.

Set **CODEARTS_CLI_AK** to the value of **Access Key Id** and **CODEARTS_CLI_SK** to the value of **Secret Access Key**.

Step 3 After the configuration is complete, right-click the project you want to launch and choose **Open Windows Terminal here**, enter **codearts**, and press **Enter**.

Enter the TUI development environment. The configuration is permanently effective.

----End

macOS

The following describes how to perform authorization by configuring process-level variables and system environment variables in macOS.

Configuring Process-Level Variables

Step 1 Open the terminal and run the following commands to grant temporary authorization for the current window:

```
export CODEARTS_CLI_AK="Access Key Id"
export CODEARTS_CLI_SK="Secret Access Key"
```

Step 2 After the commands are executed, run the following commands in the current window to check whether the configuration has taken effect:

```
echo $CODEARTS_CLI_AK
echo $CODEARTS_CLI_SK
```

If the key information is displayed, the configuration has taken effect. If no output or an empty line is displayed, the configuration has not taken effect.

----End

Configuring System Environment Variables

Step 1 Run the following commands to add environment variables to the **~/.zshrc** file:

```
echo 'export CODEARTS_CLI_AK="Access Key Id"' >> ~/.zshrc
echo 'export CODEARTS_CLI_SK="Secret Access Key"' >> ~/.zshrc
```

Step 2 Run the following command to permanently save the two variables:

```
source ~/.zshrc
```

Step 3 Run the following commands to check whether the configuration has taken effect:

```
echo $CODEARTS_CLI_AK
echo $CODEARTS_CLI_SK
```

If the key information is displayed, the configuration has taken effect. If no output or an empty line is displayed, the configuration has not taken effect.

----End

Linux

You can select an authorization method as required.

Table 2-2 Comparison of different authorization methods

Authorization Method	Effective Scope	Lifecycle	Applicable Shell	Permission Requirements	Typical Scenarios
Temporary process authorization	Current terminal window only	Expires immediately after the terminal is closed; cleared upon a server restart	bash, zsh, and sh	Common users	Temporary debugging, one-off tests, and temporary script execution, without the need to retain keys
Permanent Bash user authorization	Currently logged-in users and all new terminals	Persists permanently; remains effective after a terminal or server restart	Linux default Bash (CentOS, Ubuntu, or Kylin)	Common users	Personal servers, frequent daily use, operations only performed by the user

Temporary Process Authorization

The configuration takes effect only in the current terminal window. After the window is closed, the configuration automatically becomes invalid. This method is suitable for one-time debugging and temporary command execution. Keys are not stored permanently.

Step 1 Configure temporary environment variables (AK/SK).

Run the following commands in the terminal to inject temporary access credentials to the current session window:

```
export CODEARTS_CLI_AK="Access Key Id"
export CODEARTS_CLI_SK="Secret Access Key"
```

Replace *Access Key Id* and *Secret Access Key* with the actual values. For details about how to obtain the values, see **Preparations**.

Step 2 Check whether the authorization is successful.

For example, run the **codearts models** command in the terminal to check whether the authorization is successful.

- If information similar to the following is displayed, the authorization is successful.

```
root@szvphis32149187:~# codearts models
model_id                                model_name
-----------------------------------------------------------------
huaweicloud-maas/GLM-5.1                GLM-5.1
huaweicloud-maas/GLM-5.2                GLM-5.2
```

- If an authentication failure or error message is displayed, run the following commands to check whether the AK/SK is correct:

```
echo $CODEARTS_CLI_AK
echo $CODEARTS_CLI_SK
```

----End

Permanent Bash User Authorization

If the current login user needs to be used for a long time, you are advised to use this method for permanent authorization. Then, the configuration will be retained even if the terminal or the server is restarted. Administrator permissions are not required.

Step 1 Add the environment variables to the Bash configuration file.

Run the following commands to add the environment variables CODEARTS_CLI_AK and CODEARTS_CLI_SK to the **.bashrc** file of the current user:

```
echo 'export CODEARTS_CLI_AK="Access Key Id"' >> ~/.bashrc
echo 'export CODEARTS_CLI_SK="Secret Access Key"' >> ~/.bashrc
```

Replace *Access Key Id* and *Secret Access Key* with the actual values. For details about how to obtain the values, see **Preparations**.

Step 2 Reload the Bash configuration for the added environment variables to take effect immediately.

```
source ~/.bashrc
```

Step 3 Check whether the authorization is permanently effective.

Close the current terminal window, open a new terminal session, and run the **codearts models** command to check whether the authorization is effective.

- If the model information is displayed, the permanent authorization is effective.

```
root@szvphis32149187:~# codearts models
model_id                                model_name
-----------------------------------------------------------------
huaweicloud-maas/GLM-5.1                GLM-5.1
huaweicloud-maas/GLM-5.2                GLM-5.2
```


- If an authentication failure or error message is displayed, run the following commands to check whether the AK/SK is correct:
  ```
  echo $CODEARTS_CLI_AK
  echo $CODEARTS_CLI_SK
  ```

----End

2.4 Upgrade

CodeArts Agent CLI supports both automatic upgrade and manual upgrade.

Automatic Upgrade

When you start CodeArts Agent CLI and the current version is not the latest, CodeArts Agent CLI prompts you whether to perform an automatic upgrade.

The screenshots in this section use the Command Prompt (CMD) interfaces as examples.

Step 1 Check whether a new version is detected. If yes, you will be prompted to perform an upgrade.

Step 2 Select **Yes** to start the upgrade.

If **Installation Successful** is displayed, CodeArts Agent CLI has been successfully upgraded.

----End

Manual Upgrade

Run the **codearts upgrade** command. If "Done" is displayed in the command output, the manual upgrade is successful.

Figure 2-4 Manual upgrade

3 Permissions

You can configure the permission file to control the execution permissions of CodeArts Agent CLI. Supported permission actions include automatic execution, approval required, and execution denied.

Configuration Path

Configuration path of the permission file: ~/.codeartsdoer/cli-data/storage/permission/global.json

Configuration Format

Permission configuration format:

```
[
  {
    "permission": "Permission name",
    "pattern": "*",
    "action": "Permission policy"
  }
]
```

For details about permission names, see **Table 3-1**. For details about permission policies, see **Table 3-2**.

Permission Policy Configuration Parameters

Table 3-1 Permission names

Permission	Description
edit	Modify all files. This permission is triggered when the AI creates files, edits existing ones, or performs bulk code changes.
write	Write data to files.
deleteFile	Delete files.

Permission	Description
bash	Run shell commands and match the fully parsed commands. For example, this permission is triggered when the AI executes commands such as **git status**, **npm install**, and **python manage.py runserver**. Bash command permissions must be used together with **bash_mode**, indicating the whitelisted commands in sandbox mode. **bash_mode** also needs to be configured in the **~\.codeartsdoer\cli-data\storage\permission\config.json** file. For details about the configuration modes supported by **bash_mode**, see **Table 3-3**.
webfetch	Obtain URLs. This permission is triggered when the AI accesses external APIs, downloads files, or obtains web page content.
external_directory_read	Read paths outside the project directory. This permission is triggered when the AI reads files such as **/etc/hosts** and **~/.ssh/id_rsa** that are not in the project working directory, .
external_directory_write	Modify paths outside the project directory. This permission is triggered when the AI writes data to or modifies system or user configuration files such as **/etc** or **~/.bashrc**.
dotfile	Modify dotfiles (configuration files starting with a dot). This permission is triggered when the AI modifies hidden configuration files such as **.gitignore**, **.env**, **.vscode**, or **settings.json**.
doom_loop	Loop protection for tool calls. This permission is triggered when the same tool (for example, read) is called three consecutive times with the same input to prevent infinite loops.

Table 3-2 Permission policies

Action	Description
allow	Direct execution.
ask	Approval required. When you configure **ask**, the following options are displayed: • **Allow once**: Only the current execution is allowed. The same command will prompt for approval again the next time. • **Allow always**: For the current session, commands of this type will run without further prompts for approval. • **Reject**: The command is not executed and the session continues. This is suitable for actions that you consider risky or unnecessary.
deny	Execution denied

NOTE

- In TUI development mode, the sandbox mode is used by default. The options **Allow once**, **Allow always**, and **Reject** are returned.
- In CLI development mode, the execution is automatically denied. To enable the execution, set the action to **allow**.

Table 3-3 Modes supported by bash_mode

Mode	Description
sandbox	Execution in sandbox (default, most secure)
always_ask	Approval required for all commands
always_allow	Direct execution for all commands

Whitelist Configuration Example

Example:

```
[
 {
  "permission": "bash",
  "pattern": "git *",
  "action": "allow"
 },
 {
  "permission": "bash",
  "pattern": "npm *",
  "action": "allow"
 },
 {
  "permission": "bash",
  "pattern": "ls *",
  "action": "allow"
 }
]
```

This example contains three JSON objects and supports automatic execution of all commands starting with **git**, **npm**, and **ls**.

4 Custom Commands

In addition to built-in slash commands, file references, and Bash commands, you can also add custom commands by defining them in **.jsonc** and **.md** files. For details, see **Customizing Commands in .jsonc** and **Customizing Commands in .md**.

Customizing Commands in .jsonc

CodeArts Agent CLI allows you to configure custom commands in **codearts_cli.json** or **codearts_cli.jsonc** (JSON file with comments).

You can configure custom commands under personal-level or project-level paths. For details, see **Table 4-1**.

CAUTION

You need to manually create a file and define custom commands in it. Then, run CodeArts Agent CLI.

Table 4-1 Configuration path

Configuration Level	Description	Location
Personal-level	The personal-level configuration is the global default configuration. You can set personal preferences (such as themes or shortcuts), which will take effect for all projects.	**~/.codeartsdoer/codearts_cli.json** or **~/.codeartsdoer/codearts_cli.jsonc** The tilde (**~**) indicates the home directory of the current user. In Windows, it is equivalent to **C:\Users***Username*****. In macOS, it is equivalent to **/Users/***Username***/**. In Linux, it is equivalent to **/home/***Username***/**.
Project-level	The project-level configuration is dedicated to a specific project. **It has a higher priority than the personal-level configuration** and can overwrite it. This configuration is used to define unified rules for a project.	*Project root directory***/.codeartsdoer/codearts_cli.json** or *Project root directory***/.codeartsdoer/codearts_cli.jsonc**

Configuration Example

The following uses the custom commands **test** and **check** as examples.

Step 1 Use the project-level configuration and Windows as an example.

Create the **codearts_cli.jsonc** file in the **./.codeartsdoer** directory of the project root directory and configure the file as follows:

In the custom commands, **command name** and **template** are mandatory.

```
{
  //Command configuration
  "command": {
    // All custom commands are defined here.
    "test": {
      // command name = test. You can rename it to run, fix, or others as you like.
      "template": "This is the prompt sent to the AI after configuration. It can be a sentence or a complex instruction.",
      "description": "A brief description of the command, displayed in the TUI command list.",
      "agent": "Name of the AI agent that executes the command. This parameter is optional.",
      "model": "Model ID specified for the command. This parameter is optional and will overwrite the default model.",
      "subtask": false // Optional Boolean attribute, which is used to control whether the command runs in the sub-agent mode.
```



```
  },
  // Example of the second custom command: check
  "check": {
    "template": "Dedicated AI prompt for the run command, which is used to run the project and check code.",
    "description": "One- click code check and project execution.",
    "agent": "",
    "model": "",
    "subtask": false
  }
 }
}
```

Step 2 Check whether the custom commands have taken effect.

- In TUI development mode, enter **/test** to verify the custom command **test**. You can also enter **/check** to verify the custom command **check**.

  Figure 4-1 Example of the TUI development mode

- In the CMD, PowerShell, or terminal, run the **codearts run check** command to associate the custom command **check**. You can also run the **codearts run test** command to associate the custom command **test**.

  NOTE

  The command output may vary. Refer to the actual output for the final result.

----End

Customizing Commands in .md

Priority: If the command names are the same, the custom .md file has a higher priority than the .json file.

You can customize a Markdown file in the **commands** directory to define custom commands. **The file name is the command name**.

Table 4-2 Path of the commands folder

Level	Location
Personal-level	~/.codeartsdoer/commands The tilde (~) indicates the home directory of the current user. In Windows, it is equivalent to **C:\Users***Username*\. In macOS, it is equivalent to **/Users/***Username*/. In Linux, it is equivalent to **/home/***Username*/.
Project-level	*Project root directory*/**.codeartsdoer/commands**

Configuration Example

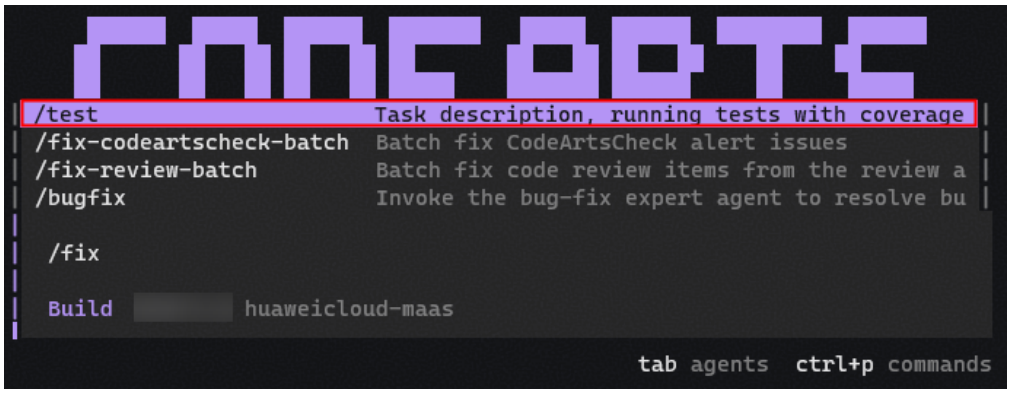
Step 1 In Windows, create **test.md** under **~\.codeartsdoer\commands** and add personal-level custom commands to the file.

```
---
description: Task description, running tests with coverage
agent: build
model: huaweicloud-maas/GLM-5.1
---
Perform a full test and generate a coverage report. Locate the failed test cases and provide fix suggestions.
```

Step 2 Restart CodeArts Agent CLI.

- In TUI development mode, enter **/test** to associate the custom command **test**.

  Figure 4-2 Example of the TUI development mode

- In the CMD, PowerShell, or terminal, run the **codearts run test** command to associate the custom command **test**.

  NOTE

  The command output may vary. Refer to the actual output for the final result.

----End

5 Sandbox

A sandbox provides an **isolated and restricted execution space** for instructions generated by agents. With strict permission control, commands run in a secure, isolated environment. This effectively blocks access to unauthorized resources (such as files and networks) and prevents high-risk commands (such as those that change permissions). For intercepted commands, agents will issue a risk prompt within a chat. These commands need your second approval before running outside the sandbox.

Constraints

Table 5-1 Constraints

Item	Description
OS	• Windows: **Windows 11 (x64) is recommended.** For Windows 10 (x64), the version must be 2019 or later, and upgrading to the latest stable version is recommended. • macOS: macOS 11 or later, compatible with ARM64 (Apple Silicon). • Linux: Huawei Cloud EulerOS 2.0, SUSE Linux Enterprise Server 12 SP5, Ubuntu 18.04/20.04/22.04/24.04 LTS, Debian 10/11/12, CentOS 8, RHEL 8/9

File Access Control

After sandbox is enabled, CodeArts Agent CLI configures file directory access permissions as described below. You can also customize the access permissions based on your service requirements. For details, see **Enabling Sandbox**.

Table 5-2 File access control

OS	Permission Type	Directory Type	Directory List
Windows	Read-only	-	All directories are readable except **critical Windows system directories and sensitive user directories**.
	Read/Write	Project directories and their subdirectories	-
	No read/No write	Critical Windows system directories	• C:\Windows\System32\config • C:\Windows\System32\drivers\etc • C:\Windows\SysWOW64\config • C:\ProgramData\Microsoft\Crypto • C:\Windows\System32\GroupPolicy • C:\Windows\System32\GroupPolicyUsers • C:\ProgramData\Microsoft\Windows\WER
		Sensitive user directories	• C:\Users*\AppData\Local\Microsoft\Credentials • C:\Users*\AppData\Roaming\Microsoft\Credentials • C:\Users*\AppData\Local\Microsoft\Protect • C:\Users*\NTUSER.DAT • C:\Users*\ntuser.dat.LOG
macOS	Read-only	-	All directories are readable except **sensitive system directories**.

OS	Permission Type	Directory Type	Directory List
	Read/Write	Workspace and additionally configured directories	• Temporary directories: /tmp, /var/folders, and TMPDIR environment variable paths • Cache directories: ~/Library/Caches and ~/.cache • Dependency directories of common tools: ~/.local/lib, ~/.local/bin, and ~/.local/share • Toolchain and dependency directories of common development languages (Go, Java, Python, Node.js, Rust, and Ruby)
	No read/No write	Documents/ Desktop/ Downloads (privacy-related)	• ~/Desktop • ~/Documents • ~/Downloads • ~/Pictures • ~/Movies • ~/.ssh • ~/.zsh_history
		Password/ Wallet/ Keychain-related	~/Library/Keychains
		System-level sensitive configuration	• /etc/passwd • /private/etc/passwd • /etc/group • /private/etc/group • /etc/hosts • /private/etc/hosts • /etc/resolv.conf • /private/etc/resolv.conf • /etc/pam.d • /private/etc/pam.d
Linux	Read-only	-	All directories are readable except **sensitive system directories**.

OS	Permission Type	Directory Type	Directory List
	Read/Write	Workspace and additionally configured directories	• Temporary directories: /tmp and TMPDIR environment variable paths • Cache directories: ~/.cache and XDG_CACHE_HOME environment variable paths • Dependency directories of common tools: ~/.local/lib, ~/.local/bin, and ~/.local/share • Toolchain and dependency directories of common development languages (Go, Java, Python, Node.js, Rust, and Ruby)
	No read/No write	Critical Linux system directories	• /etc/shadow • /etc/passwd • /etc/group • /etc/gshadow • /etc/resolv.conf • /etc/sudoers

Enabling Sandbox

The following describes how to enable sandbox in the TUI development environment. In the CLI development environment, run the **codearts run "*prompt*" --sandbox** command.

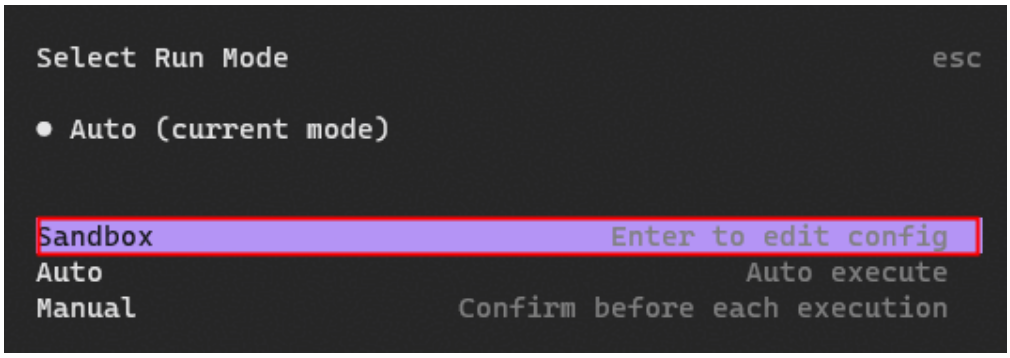
Step 1 Enter the TUI development environment.

1. Open the root directory of the target project.
2. Right-click in the blank area and select **Open Windows Terminal here**.
3. Enter **/codearts** in the terminal and press **Enter** to enter the TUI development environment.

Step 2 Enable the sandbox mode.

1. In the TUI dialog box, enter **/run-mode** and press **Enter**. The **Select Run Mode** configuration is displayed.
2. Click **Sandbox** to go to the **Sandbox Config** page.

Figure 5-1 Selecting the Sandbox mode



Step 3 Configure the command whitelist.

Add prefixes of specific commands to the whitelist as required. Commands added to the whitelist bypass the sandbox mechanism and are executed outside the sandbox.

Step 4 Select a network policy.

Table 5-3 Network policy description

Parameter	Description
Allow All	Allows access to all internal and external network resources.
Local Network Only	Only allows access to local networks (LAN/intranet); blocks all external internet access.
Block All	Blocks all network connections and prohibits access to any internal or external resources.
Custom Policy	Allows you to modify the JSON policy configuration file to customize the file and network access scope for processes within the project sandbox environment. For details, see **Step 5**.

Step 5 Customize a network policy.

1. In the **Network Policy** area, select **Custom Policy** and click **Confirm**. The **Custom Policy** page is displayed.
2. In the JSON policy configuration file, modify the configuration as required.

   The initial structure of the file is as follows:

```
{
  "filesystem": {
    "readWrite": [],
    "readOnly": []
  },
  "network": {
    "default": "Allow",
    "allow": [],
    "deny": []
  },
  "resources": {
    "cpu": 50,
```



```
    "memory": 8
  }
}
```

Table 5-4 Parameters in the JSON policy configuration file

Parameter	Mandatory	Type	Description
filesystem	No	**filesystem** Object	Used to precisely control the sandbox's access permissions to the local file system. If it is not set (the filesystem field is empty or does not exist), the sandbox's built-in file system security policy will be applied.
network	No	**network** Object	Used to control the network access policies for processes within the sandbox, supporting configurations to allow or block access to specific network resources. If it is not set, network access is allowed by default.
resources	No	**resources** Object	Used to define the maximum limit of computing resources during sandbox runtime, ensuring service stability and preventing resource abuse. If it is not set, the system will share the host machine's resources.

Table 5-5 Fields in filesystem

Parameter	Type	Default Value	Format of Supported Paths	Priority	Description
readWrite	Array	[ ]	– Absolute path: for example, **/home/user/project** or **C:\Projects** – Relative path: for example, **./src** or **./config** – Environment variable: $HOME (Linux/Mac) or %USERPROFILE% (Windows) – Abbreviation of the home directory: ~	readOnly > readWrite > Default system policy If a specific path matches both readOnly and readWrite, readOnly shall be applied.	List of read/write paths.
readOnly	Array	[ ]			List of read-only paths.

Table 5-6 Fields in network

Parameter	Type	Default Value	Priority	Description
default	String	Allow	deny > allow > default If both allow and deny are configured, deny shall be applied.	Default network policy. – **Allow**: access allowed. – **Deny**: access denied. **NOTE** This field supports two configuration formats: *Domain*:*Port* and *IP address*:*Port*. Wildcards are supported for the domain part, and CIDR notation is supported for IP addresses. Multiple ports can be separated with commas (,). If no port is specified, the policy applies to all ports by default.
allow	Array	[ ]		List of network rules that allow access.
deny	Array	[ ]		List of network rules that deny access.

Table 5-7 Fields in resources

Parameter	Type	Default Value	Min. Value	Description
cpu	Integer	50	20	CPU usage, in percentage.
memory	Integer	8	1	Memory size, in GB.

The following is an example of the JSON policy configuration file:

```
{
  "filesystem": {
```



```
        "readWrite": [
            "/home/user/project/output",
            "~/workspace/temp"
        ],
        "readOnly": [
            "/etc/systemd",
            "%USERPROFILE%/.ssh"
        ]
    },
    "network": {
        "default": "Allow",
        "deny": [
            "10.0.0.0/8",
            "192.168.0.0/16"
        ]
    },
    "resources": {
        "cpu": 50,
        "memory": 8
    }
}
```

3. Click **Confirm** to exit the current setting page and complete the enabling of the sandbox mode.

----End

Execution Policy for High-Risk Commands

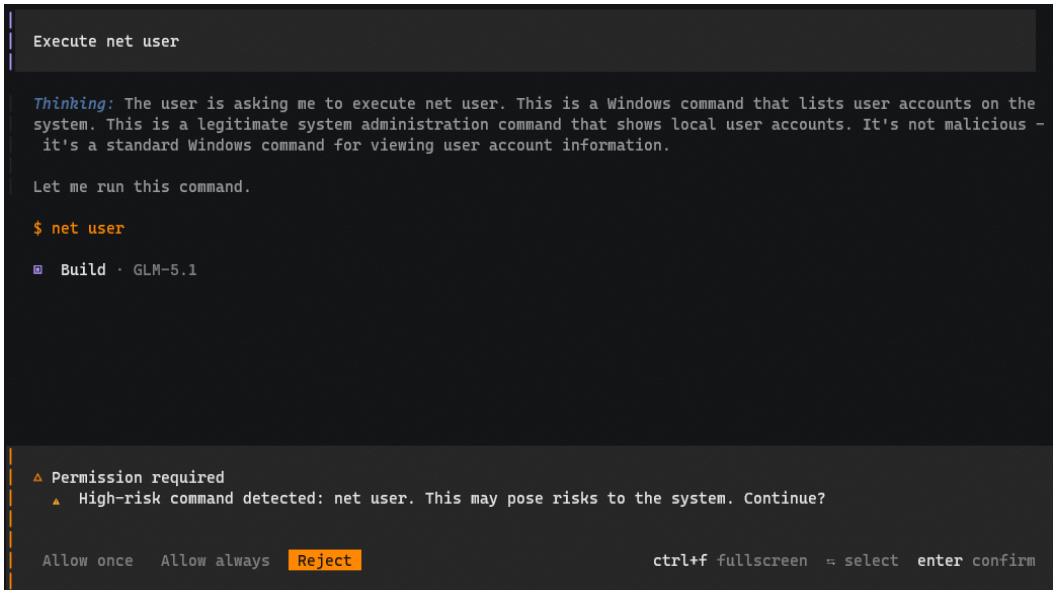
When the agent detects a high-risk command, the AI will send a prompt in the chat session. (**In the TUI development environment, a dialog box is displayed to ask whether the command is intercepted. In the CLI development environment, the command is intercepted directly.**) You need to evaluate risks and select an execution mode as required.

- **Allow once**: Only the current execution is allowed. The same command will prompt for approval again the next time.
- **Allow always**: For the current session, commands of this type will run without further prompts for approval.
- **Reject**: The command is not executed and the session continues. This is suitable for actions that you consider risky or unnecessary.

NOTE

To disable sandbox, see **Disabling Sandbox**.

Figure 5-2 Example of high-risk command interception



Disabling Sandbox

The following describes how to disable sandbox in the TUI development environment. In the CLI development environment, run the **codearts run "*prompt*" --auto** command.

Step 1 Enter the TUI development environment.

1. Open the root directory of the target project.
2. Right-click in the blank area and select **Open Windows Terminal here**.
3. Enter **/codearts** in the terminal and press **Enter** to enter the TUI development environment.

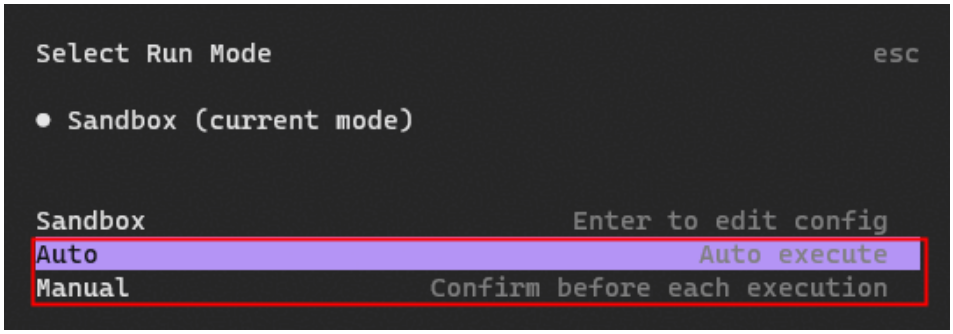
Step 2 Disable the sandbox mode.

Disabling sandbox is to set the run mode to **Auto** or **Manual**.

1. In the TUI dialog box, enter **/run-mode** and press **Enter**. The **Select Run Mode** configuration is displayed.
2. Select **Auto** or **Manual** and press **Enter** to exit the sandbox run mode.
   - **Auto**: The agent directly executes all commands without your approval.

     For security purposes, you are advised to select Auto only when necessary. In this mode, the agent bypasses all security checks and may perform high-risk operations without prior notification.
   - **Manual**: Before executing any command, the agent sends a confirmation prompt to you. The command proceeds only after you manually confirm it.

Figure 5-3 Switching the sandbox run mode



----End

6 Agents

Agents are programming assistants tailored for diverse development scenarios. CodeArts Agent CLI provides multiple out-of-the-box agents that require no manual creation, aiming to provide efficient programming assistance for various development scenarios. In addition to built-in agents, you can also create custom agents.

Built-in Agents

Built-in agents are classified into **primary agents** and **sub-agents**. Primary agents are the primary assistants you interact with. You can use the **Tab** key to switch between them. These agents handle your main conversations. Sub-agents are specialized assistants that primary agents can call to perform specific tasks. You can manually call a sub-agent by entering an at sign (@) followed by the sub-agent name in a session.

Table 6-1 Built-in agents

Type	Creator	Description
Primary agent	System built-in	CodeArts Agent CLI has two core primary agents built in: Build and Plan. They are responsible for development and planning, respectively. • Primary agent: Build The default development execution agent, which has full access to tools. It is suitable for scenarios where actual development operations are required, such as modifying code, executing commands, or creating files. Typical scenarios: – Writing, modifying, or deleting code files – Executing terminal commands for build, test, and deployment – Creating new functional modules or fixing bugs – Performing any actual operations that implements changes to the project • Primary agent: Plan A constrained read-only agent that focuses on analyzing code structure, designing solutions, and formulating implementation plans. **It does not make any substantial changes to the code repository**. Typical scenarios: – Analyzing code structure and logic – Designing functional solutions or refactoring strategies – Locating bugs and planning fixes – Architecture review and feasibility analysis – Any scenario where **you need to think clearly before taking action**

Type	Creator	Description
Sub-agent	System built-in	Sub-agents are specialized assistants that primary agents can call to perform specific tasks. You can manually call a sub-agent by entering an at sign (@) followed by the sub-agent name in a session. **NOTE** You can view the built-in sub-agents using the following commands: • CLI environment: **codearts run "***<natural-language-description>***"**. The description must contain the **subagent** keyword. • TUI environment: Enter an at sign (@) in the text box. • developer-test-agent: An agent used to generate, fix, and optimize unit tests, and review real code repositories. • explore: A code repository exploration agent dedicated to quickly locating files, searching code keywords, and answering related questions. • general: A general-purpose agent used to research complex problems and execute multi-step tasks. • rule-generator: A rule generator. • spec-design-agent: An agent used to generate comprehensive technical designs, converting requirements (what to do) into architectures (how to do it). • spec-requirement-agent: An agent used to generate requirements for the Easy Approach to Requirements Syntax (EARS) format based on the project description and context. • spec-task-agent: An agent used to generate implementation tasks based on requirements and design.

Custom Agents

In addition to built-in agents, CodeArts Agent CLI also supports custom agents. **When project-level and personal-level agents share the same name, the project-level agent takes precedence over the personal-level agent.**

Table 6-2 Classification of custom agents

Type	Scope	Description
Custom primary agent Custom sub-agent	Project-level	Effective only for the agents of the current project, distributed along with the repository, and stored locally. Storage location: *Project root directory*/**.codeartsdoer/agents**
	Personal-level	Agents are based on personal habits or preferences and take effect only for all projects of the current user. Storage location: **~/.codeartsdoer/agents** The tilde (**~**) indicates the home directory of the current user. In Windows, it is equivalent to **C:\Users***Username*****. In macOS, it is equivalent to **/Users/***Username***/**. In Linux, it is equivalent to **/home/***Username***/**.

Creating Agents Using Markdown Files

CodeArts Agent CLI allows you to create local project-level or personal-level agents through Markdown files. **When project-level and personal-level agents share the same name, the project-level agent takes precedence over the personal-level agent.**

Custom Agent File Format

Custom agents are written in the Markdown file format. The file header configures metadata using YAML front matter, and the body contains the system prompt and role instructions.

```
---
description: Agent desc # Agent function description
mode: subagent          # Agent description
model: provider/model     # If the model is configured in this format, the default model will be overwritten.
tools:                  # Tool permission configuration.
  tool1: true           # Allows the use of tool1.
  tool2: false            # Prohibits the use of tool2.
---

# Prompt
Here lists the system prompt content of the agent.
```

The following table describes the parameters of the custom agent.

Table 6-3 Markdown file parameters

Parameter	Mandatory	Type	Default Value	Description
description	Yes	String	N/A	Description of the custom agent

Parameter	Mandatory	Type	Default Value	Description
mode	Yes	String	subagent	Role of the agent • **subagent**: The agent can only be called as a sub-agent. • **primary**: The agent can only be a primary agent and cannot be called. • **all**: The agent can be a primary agent or called as a sub-agent.
model	No	String	N/A	Format: provider/model
tools	No	**Object**	N/A	Tool permissions available to the agent. For details about the tools supported by CodeArts Agent CLI, see **Table 6-5**. You can also use * to match all tools. Example: tools: "*":false # Prohibits the use of all tools.
hidden	No	Boolean	N/A	Visibility control If this parameter is set to **true**, the agent is hidden from the @ autocomplete menu. This setting is intended for internal agents that are called only through programmatic calls.
disable	No	Boolean	N/A	Switch control • **true**: disables the agent. • **false**: enables the agent.
Prompt	Yes	String	N/A	System prompt of the custom agent

Table 6-4 Object value description for tools

Parameter	Description
true	Allows the use of a tool.
false	Prohibits the use of a tool.

Table 6-5 Tools supported by CodeArts Agent CLI

Tool	Description
read	Reading files
write	Writing data to files
edit	Editing files
bash	Running commands
glob	File pattern matching
grep	Retrieving content
task	Sub-task
webfetch	Obtaining webpage content
question	Inquiring users

Example Custom Agent

This example describes how to configure a personal sub-agent in Windows.

Step 1 Go to the **~/.codeartsdoer/agents** directory, create the **test.md** file, and write the content below to the file.

The tilde (~) indicates the home directory of the current user. In Windows, it is equivalent to **C:\Users***Username*****. In macOS, it is equivalent to **/Users/***Username***/**. In Linux, it is equivalent to **/home/***Username***/**.

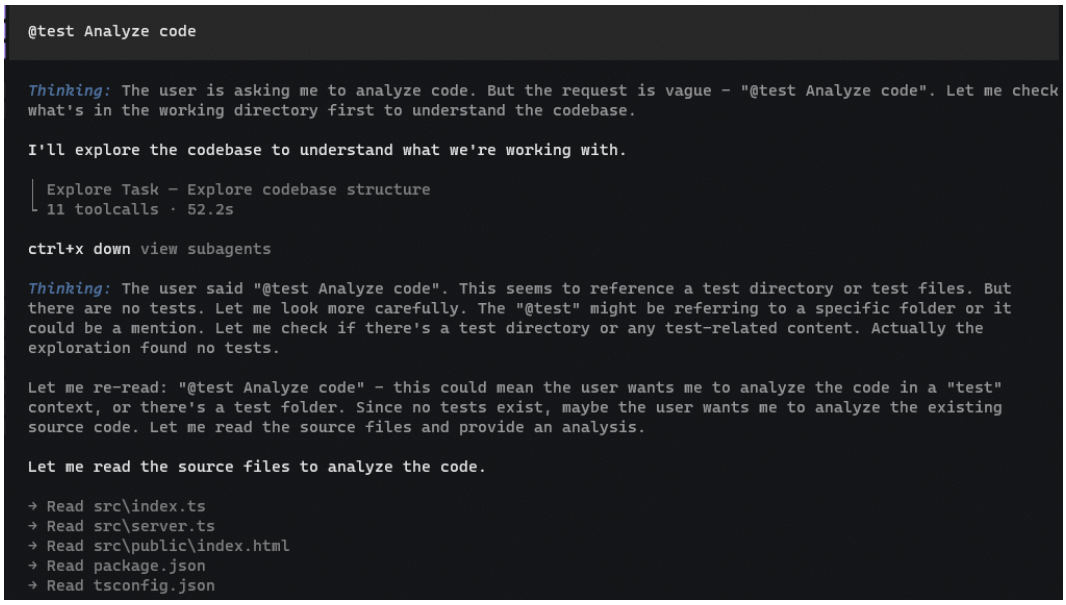
```
---
# ========== Mandatory ==========
description: (Mandatory) Description of the agent's functions and application scenarios. The primary agent determines when to call this agent based on this description.
mode: subagent            # Enumerated values: subagent (sub-agent) / primary (primary agent) / all (both).

# ========== Model Configuration ==========
model: provider/model-id # Custom model. Format: provider/model-id. You can run the /models command to view the supported models.

# ========== Behavior Control ==========
hidden: false             # true: Hide from the @ autocomplete menu, only available for internal programmatic calls.
disable: false            # true: Disable the current agent.

# ========== Tool Permissions ==========
tools:
  write: false            # File write permission.
  edit: false             # File edit permission.
  bash: false             # System command execution permission.
  read: false             # File read permission.

# System prompt (supports multi-line Markdown format)
You are a senior code review expert. Focus on:

- Code quality and best practices
- Potential bugs and edge cases
- Performance impact
- Security risks

Provide constructive feedback and do not directly modify the code.
```


Step 2 Go to any code project and run the following command:

```
@test Analyze code
```

After the command is delivered, the AI specifies a custom sub-agent to analyze code.

Figure 6-1 Calling the test sub-agent to analyze code



----End

7 Skills

Configuration Directory

Skills can be classified into project-level and user-level skills based on their application scope.

Type	Scope	Storage Path
Project	Applied to the current project and stored in a local path	*Project root directory*/**.codeartsdoer/skills**
User	Applied to all projects of the current user	**~/.codeartsdoer/skills** The tilde (**~**) indicates the home directory of the current user. In Windows, it is equivalent to **C:\Users***Username*****. In macOS, it is equivalent to **/Users/***Username***/**. In Linux, it is equivalent to **/home/***Username***/**.

If a project-level skill and a user-level skill share the same name, the project-level skill takes precedence.

Skill Directory Structure

A skill is a folder with a necessary **SKILL.md** file and optional resources. It bundles the knowledge and tools needed to perform particular tasks.

```
skill-name/
├── SKILL.md (Required)
│   ├── YAML frontmatter metadata (Required)
│   │   ├── name: (Required)
│   │   └── description: (Required)
│   └── Markdown instructions (Required)
└── Bundled Resources/ (Optional)
    ├── scripts/      - Executable code (Python, Bash, etc.)
    ├── references/   - Documents to be loaded into the context
    └── assets/       - Files used in the output (templates, icons, fonts, etc.)
```


Table 7-1 Skill parameters

Parameter	Description
skill-name	Serves as the root directory for the entire skill. Replace *skill-name* with your specific skill's name, such as **data-analysis-skill**. The skill name consists of lowercase letters, digits, and hyphens (-). It cannot start or end with a hyphen, cannot contain consecutive hyphens, and must be 1 to 64 characters in length.
SKILL.md	Serves as the core description file of a skill. Its name cannot be modified. It includes the following essential sections: • **YAML frontmatter metadata**: Placed at the very top of the **SKILL.md** file and enclosed by **---** at the beginning and end, this section provides structured information readable by AI. It contains two fields: name and description. – name: The skill name consists of lowercase letters, digits, and hyphens (-). It cannot start or end with a hyphen, cannot contain consecutive hyphens, and must be 1 to 64 characters in length. **The value of name must be the same as that of skill-name.** – description: The skill description contains a maximum of 1,024 characters. This field is critical for CodeArts Agent to recognize the skill's usage scenarios, and must clearly describe the skill functions and trigger conditions. • **Markdown instructions**: The Markdown content below the YAML metadata, used to detail the usage methods, parameters, examples, and dependencies of the skill.
Bundled Resources (optional)	Holds files needed for the skill. While not mandatory, it enhances the skill's functionality. • **scripts/**: contains executable files like Python and shell scripts that perform specific tasks for the skill. • **references/**: holds essential documents like APIs, domain knowledge, and company policies that support agent skills. It acts as the agent's **knowledge base**. Documents are loaded into the context when needed for skill execution. • **assets/**: contains resource files like fonts, company logos, and PowerPoint templates. The agent uses these files directly in its output without loading them into the context window.

Example

This document uses Windows as an example to describe how to create a user-level skill. To be specific, create a **test** folder in the **.codeartsdoer/skills** directory, create a **SKILL.md** file in the folder, and write the following content into the file:

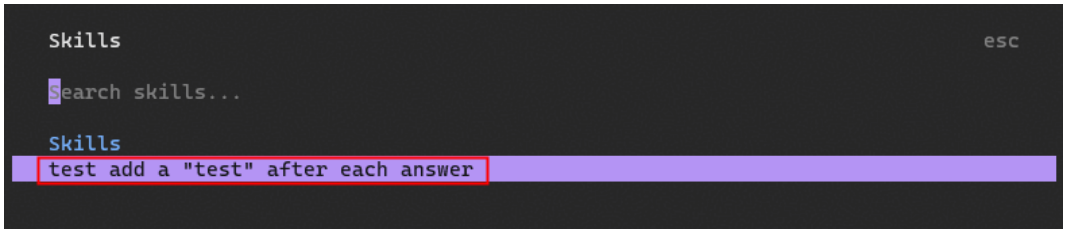

```
---
name: test
description: add a "test" after each answer
---
#Add a suffix to each answer.
Add "test" to the end of each answer.
```

Verify the skill by starting it via **TUI** or **CLI**.

Starting a Skill Using TUI

Step 1 Enter the TUI development mode and enter **/skills** to view skills.

Figure 7-1 Viewing skills



Step 2 Select the **test** skill, enter the **test** command, and press **Enter** to apply the **text** skill.

----End

Starting a Skill Using CLI

Step 1 Open CMD and enter the following command:

```
codearts run add a "test" after each answer
```

Step 2 If the following information is displayed, the **test** skill has taken effect:

```
C:\Users\Username>codearts run add a "test" after each answer

> build · GLM-5.1

→ Skill "test"

The skill has been loaded. From now on, I will add "test" at the end of each answer.
```

----End

8 Hooks

Hooks are the core mechanism of the CodeArts Agent CLI plugin system. They allow developers to inject custom logic during key phases such as tool execution, message processing, and permissions control. A plugin declares the lifecycle events it is interested in by returning a hook object. When an event is triggered, the custom function of the plugin is called.

Storage Paths for Plug-ins and Dependency Files

A plug-in is a JavaScript/TypeScript module that operates by exporting plug-in functions. Each function receives a context object and returns the corresponding hook object. If external packages are required within a plug-in file, you must create a **package.json** file in the configuration directory and configure the necessary dependencies. CodeArts Agent automatically installs these dependencies when it starts.

Table 8-1 Plug-in file loading paths

Plug-in Type	Plug-in File Storage Path	Plug-in Dependency Storage Path	Description
Project	Root directory of the current project: **./codeartsdoer/plugin** or **./codeartsdoer/plugins**	Root directory of the current project: **./codeartsdoer/**	Valid only for the current project.
User	Local **%USERPROFILE%/.codeartsdoer/plugin** or **%USERPROFILE%/.codeartsdoer/plugins**	Local **%USERPROFILE%/.codeartsdoer/**	Valid for all projects of the current user.

Supported Hook Events

For details about the hook events supported by CodeArts Agent CLI, see **Table 8-2**. For details about the examples of each hook event, see **Hook Event Reference**.

Table 8-2 Supported hook events

Event Category	Event Name	Trigger Time	Blockable	Typical Application Scenario	Description
Chat message	chat.message	Triggered after a user sends a message but before it enters the processing workflow.	Yes	Chat message processing	Automatically triggered when a new message arrives. Supports editing message content and parts.
Chat parameter	chat.params	Triggered before each LLM call.	Yes	Chat parameter modification	Supports dynamically adjusting LLM inference parameters (including temperature, topP, topK, etc.) to optimize model output as needed.
Chat request header	chat.headers	Triggered before each LLM call.	Yes	Chat request header modification	Supports customizing and modifying HTTP request header fields to adapt to complex requirements such as network proxies and interface authentication.

Event Category	Event Name	Trigger Time	Blockable	Typical Application Scenario	Description
Chat response	chat.response	Triggered after the LLM returns a response.	No	Chat response logging	Automatically captures core metrics of the model response, such as tokens, cost, and duration, achieving precise billing statistics, efficient troubleshooting, and complete log retention.
Chat error	chat.error	Triggered when an error occurs during an LLM call.	No	Chat error logging	Logs error details when an LLM call fails.
Chat compression	chat.compression	Triggered during context compression.	No	Chat compression logging	Logs the token counts before and after context compression to assist in storage and performance tuning.
Chat end	chat.finished	Triggered when a chat ends.	No	Chat end logging	Logs the chat end event, including the token consumption and running information of the current run.

Event Category	Event Name	Trigger Time	Blockable	Typical Application Scenario	Description
User approval	user.approval	Triggered when a user confirmation or approval action is required.	Yes	User approval logging	Intercepts high-risk operations and initiates manual confirmation and approval, controlling the execution permissions of sensitive operations.
Turn end	turn.end	Triggered when each chat turn ends.	No	Turn end logging	Logs the end of each chat turn, containing key information such as processing duration and termination reasons.
Command execution	command.execute.before	Triggered before a command is executed.	Yes	Command preprocessing	Supports custom configuration of parts to achieve command preprocessing and flexible adaptation.
Tool execution	tool.execute.before	Triggered before a tool is executed.	Yes	Tool execution pre-parameter adjustment	Supports customizing and modifying args before tool execution to achieve validation, dynamic rewriting, and service adaptation.

Event Category	Event Name	Trigger Time	Blockable	Typical Application Scenario	Description
	tool.execute.after	Triggered after a tool is executed.	Yes	Tool execution post-result processing	Supports customizing and modifying returned results, including the title, output, and metadata fields.
Tool definition	tool.definition	Triggered before a tool definition is sent to the LLM.	Yes	Tool definition modification	Supports customizing configuration descriptions and parameters before tool definitions are pushed to the model, ensuring precise adaptation to service scenarios.
Shell environment	shell.env	Triggered when fetching Shell environment variables.	Yes	Shell environment variable configuration and management	Supports adding or modifying environment variables when Shell environment variables are fetched.
Permission request	permission.ask	Triggered when permission verification is required.	Yes	Permission request processing	Processes permission validation requests and supports configuring three strategies: allow, deny, and ask.

Event Category	Event Name	Trigger Time	Blockable	Typical Application Scenario	Description
Message conversion	experimental.chat.messages.transform	Triggered before the message list is sent to the LLM.	Yes	Message conversion	Supports preprocessing the chat list before it is reported to the model.
System message conversion	experimental.chat.system.transform	Triggered before the system prompt is sent.	Yes	System message conversion	Supports optimizing prompt content before the prompt is sent.
Session compression	experimental.session.compacting	Triggered before context compression starts.	Yes	Chat compression	Supports customizing the compression prompt before context compression is executed.
Text completion	experimental.text.complete	Triggered when text completion finishes.	Yes	Text completion result customization	Supports modifying the output content after text completion is completed.
Configuration loading	config	Triggered when application startup configurations are loaded.	No	Configuration loading	Performs configuration initialization and dynamic parameter injection and is triggered during the loading of custom configurations.
Common event	event	Triggered when all Bus events are subscribed to.	No	Event processing	Processes common events.

Writing an Example

This section describes how to create a user-level hook.

The following example shows how to create a JsonFormatPlugin. The plugin can automatically format a compressed JSON file in which the content is squeezed in one line. After that, the content in the file is indented by two spaces.

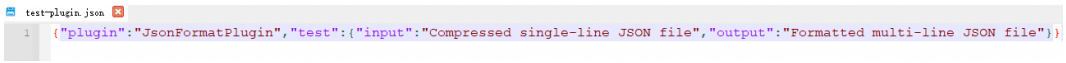
Step 1 Go to the **~\.codeartsdoer\plugin** directory and create and edit the **JsonFormatPlugin.ts** file.

```
// Import the plugin type to define the plugin.
import type { Plugin } from "@opencode-ai/plugin"
// Import the file system module to read and write disk files.
import { readFileSync, writeFileSync } from "fs"

// Export the plugin implementation. Plugin is an asynchronous function that receives plugin input parameters and returns a hook object.
export const JsonFormatPlugin: Plugin = async ({}) => {
  return {
    // Listen to the hook after the tool is executed. This function is triggered after the tool execution is complete.
    "tool.execute.after": async (
      // Input parameters include the tool name, session ID, call ID, and other parameters.
      input: { tool: string; sessionID: string; callID: string; args: any },
      // Output parameters include the title, output, and metadata of the tool execution result.
      output: { title: string; output: string; metadata: any },
    ) => {
      if (input.tool !== "read") return

      // Obtain the file path. Only .json files are processed.
      const filePath = input.args?.filePath || input.args?.[0] || ""
      if (!filePath.endsWith(".json")) return

      try {
        const raw = readFileSync(filePath, "utf-8").trim()
        const parsed = JSON.parse(raw)
        const formatted = JSON.stringify(parsed, null, 2) + "\n"
        if (formatted === raw) return

        // Write the formatted content back to the disk file.
        writeFileSync(filePath, formatted, "utf-8")
      } catch {}
    },
  }
}
```

Step 2 Go to the project folder, create and compile a compressed test file named **test-plugin.json**.

```
{"plugin":"JsonFormatPlugin","test":{"input":"Compressed single-line JSON file","output":"Formatted multi-line JSON file"}}
```

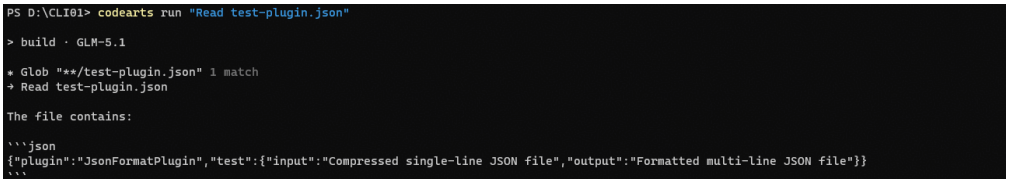
Figure 8-1 Before formatting



Step 3 Run the required command based on the development mode:

- In TUI development mode, enter the following command in the TUI dialog box and press **Enter**:
  ```
  Read test-plugin.json
  ```
- In CLI development mode, run the following command:
  ```
  codearts run "Read test-plugin.json"
  ```

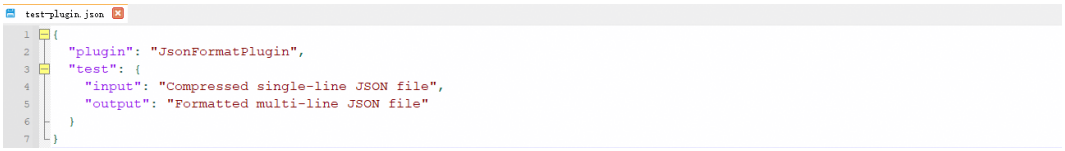

Figure 8-2 Command in CLI development mode



Step 4 After the command is executed, the **test-plugin.json** file is automatically formatted with proper line breaks and indentation. When the same file is read again, it will not be processed again because it has already been formatted.

Open the target JSON file. You will find that the JSON file has been formatted.

Figure 8-3 Formatted file



----End

9 Repository Indexing

CodeArts Agent provides a repository index function to help you quickly search and retrieve code-related information. This function supports creating both local user indexes and cloud-shared indexes, flexibly adapting to the requirements of different scenarios.

Constraints

Table 9-1 Constraints

Category	Description
Language	Java, JavaScript, TypeScript, and Go are supported.
Total number of code files	• Basic edition: Up to **50,000** code files can be indexed. • Professional edition: Up to **100,000** code files can be indexed.

Differences Between Local User Indexes and Cloud Indexes

Table 9-2 Differences between local user indexes and cloud indexes

Dimension	Local User Index	Cloud Index
Used by	Current user	• Enterprise: available to all enterprise members and created by the enterprise administrator, team administrator, enterprise members, or team members. • Team: available only to team members and created by the enterprise administrator, team administrator, or team members. • User: available only to members themselves.
Repository type	Unlimited	Git and GitHub repositories are supported.

Repository Indexing Commands

The following table lists the commands for managing the lifecycle of repository indexes.

Table 9-3 Repository indexing commands

Command	Description
codearts codebase --init	Creates repository indexes. Run this command in the project directory to create repository indexes.
codearts codebase --detail	Queries the repository indexing progress.
codearts codebase --update	Incrementally updates repository indexes.
codearts codebase --delete	Deletes repository indexes.

Creating Repository Indexes

You can create local user indexes and cloud indexes.

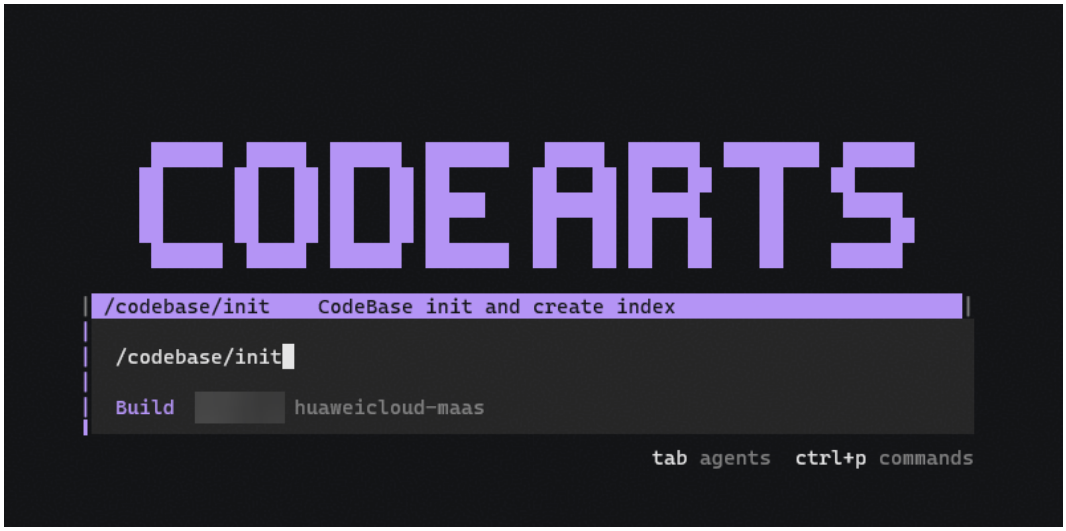
Creating Local User Indexes

Step 1 Enter the TUI development environment.

1. Open the root directory of the target project.
2. Right-click in the blank area and select **Open Windows Terminal here**.
3. Enter **/codearts** in the terminal and press **Enter** to enter the TUI development environment.

Step 2 **Create codebase indexes.** In the dialog box, enter **/codebase/init** and press **Enter** to start creating indexes.

Figure 9-1 Entering the index creation command



Step 3 **View the index creation progress.** In the dialog box, enter **/codebase/detail** and press **Enter** to view the index creation progress.

Figure 9-2 Viewing the index creation progress

Step 4 **Incrementally update repository indexes.** In the dialog box, enter **/codebase/update** and press **Enter** to index added or modified content.

The system compares the current status with the result of the last incremental update, indexes the added or modified code files, and skips unchanged files to save time.

Step 5 **Delete codebase indexes**. In the dialog box, enter **/codebase/delete** and press **Enter** to delete all indexes.

----End

Creating Cloud Indexes

Step 1 Log in to the **CodeArts Agent console**.

Step 2 In the navigation pane on the left, choose **Agent Settings** > **Repository Index**.

Step 3 Click **Create Index**.

Step 4 Set basic index information by referring to **Table 9-4** and click **Next**.

Table 9-4 Repository index parameters

Parameter	Description
Visible To	Visible scope of the repository index. • User: available only to members themselves. • Team: available only to team members and created by the enterprise administrator, team administrator, or team members. • Enterprise: available to all enterprise members and created by the enterprise administrator, team administrator, enterprise members, or team members.
Teams	This parameter is displayed only when **Visible To** is set to **Team**. Select a team from the drop-down list; only the teams which the current account belongs to are displayed.
Code Source	Source of the code repository that you want to index. • Git: a general Git repository. • GitHub: a GitHub repository, which can be a public repository or an authorized private repository.
Authorization Method	Method to authorize the system to access your code source. • **Authorize by Service**: Select a service endpoint from the drop-down list. The endpoint corresponds to the connection information of the GitHub or Git repository. If no service endpoint has been created, click **Add** to create one by referring to **Table 9-5**. • **Authorize by Personal Token**: Authorize access requests using an access token generated from your personal account.
Default Branch	This parameter is displayed only when **Code Source** is set to **Git**. Enter the branch name of a Git repository. The branch name can contain a maximum of 200 characters.

Figure 9-3 Creating a service endpoint

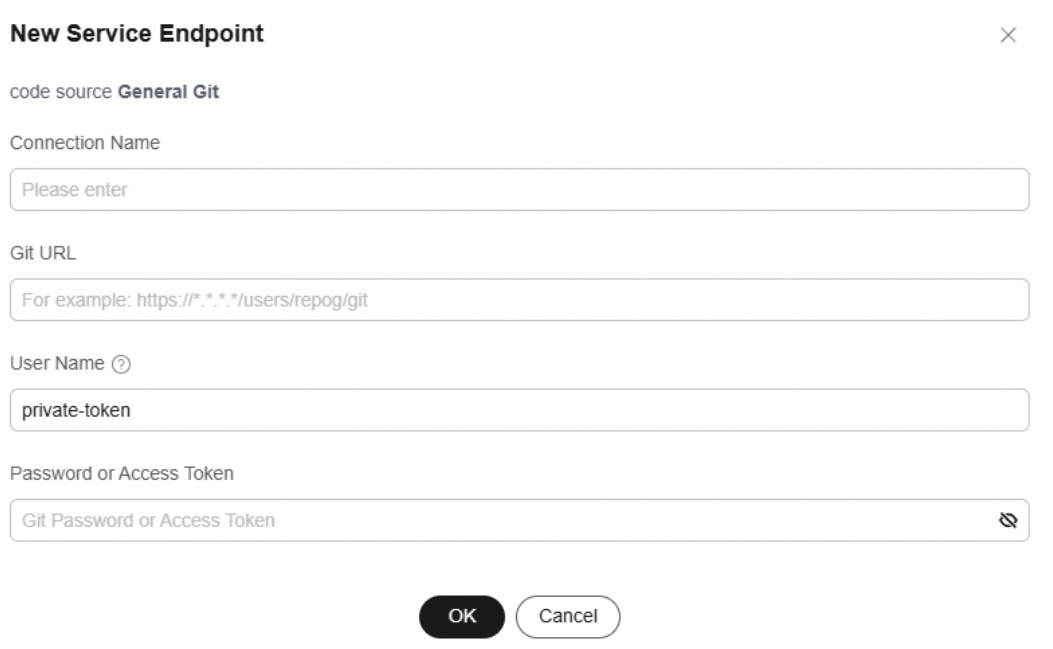


Table 9-5 Service endpoint parameters

Parameter	Description
Connection Name	Name of the custom repository to be connected. Naming rules: The name can contain letters, digits, underscores (_), and hyphens (-), but cannot start with an underscore.
Git URL	This parameter is displayed only when **Code Source** is set to **Git**. HTTPS access address of the Git repository
User Name	This parameter is displayed only when **Code Source** is set to **Git**. User name for HTTPS authentication of the Git repository.
Password or Access Token	This parameter is displayed only when **Code Source** is set to **Git**. Password or access token for HTTPS authentication of the Git repository.
Access Token	This parameter is displayed only when **Code Source** is set to **Github**. Access token for HTTPS authentication of the GitHub repository.

Step 5 Select the repository and language, and click **OK** to finish creating the index.

On the repository index page, view the newly created repository index. When its status changes from parsing to parsing completed, the repository index has been successfully parsed.

Step 6 View the index creation progress.

1. In the root directory of the **Git/Github** repository project, right-click in the blank area and select **Open Windows Terminal here**.
2. Enter **/codearts** and press **Enter** to enter the TUI development environment. The system automatically downloads team repository indexes from the cloud.
3. In the TUI dialog box, enter **/codebase/detail** and press **Enter** to view the index creation progress.

----End

10 LSP

Language Server Protocol (LSP) is a standardized protocol for providing language services. By decoupling language services from the IDE, LSP enables an independent language server process to deliver core capabilities such as code analysis, fast navigation, and symbol search.

CodeArts Agent features a built-in local LSP function. Once enabled, the agent can leverage these language services to provide code analysis, code navigation, and symbol information, enabling more accurate code comprehension and modification.

Supported Mainstream LSP Servers

Table 10-1 Supported mainstream LSP servers

LSP Server	Extension Name	Requirement
jdtls	.java	Ensure that Java SDK 21 or later has been installed.
typescript-language-server	.ts, .tsx, .js, .jsx, .mjs, .cjs, .mts, and .cts	Ensure that TypeScript has been installed.
pyright	.py, and .pyi	Ensure that Pyright has been installed.
gopls	.go	Ensure that the Go language environment and go commands are available and configured in the system environment variables.

Enabling Local LSP

Step 1 Enter the TUI development environment.

1. Open the root directory of the target project.
2. Right-click in the blank area and select **Open Windows Terminal here**.

3. Enter **/codearts** in the terminal and press **Enter** to enter the TUI development environment.

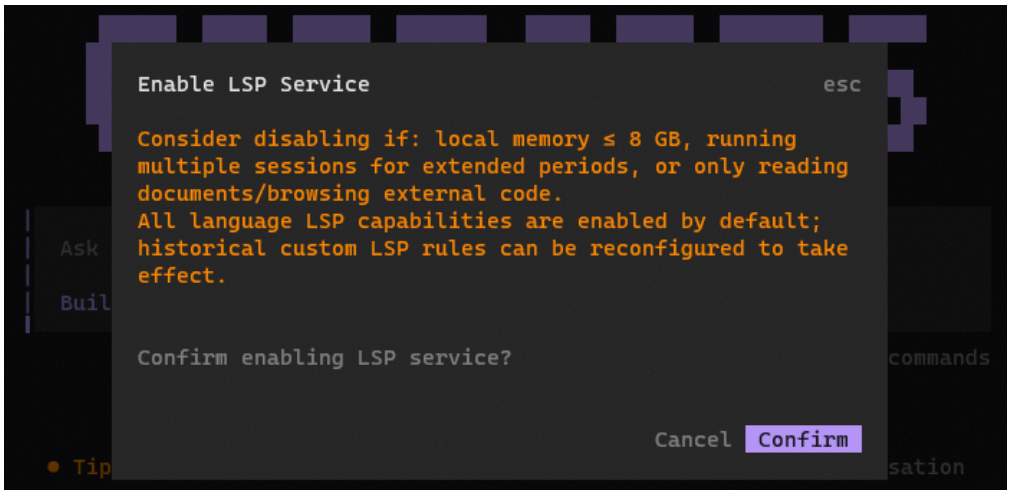
Step 2 Enable local LSP.

1. In the TUI dialog box, enter **/lsp-toggle** and press **Enter** to access the **LSP Service Switch** page.

Figure 10-1 LSP Service Switch page

2. Press **Enter** and click **OK** to enable the LSP service.

Figure 10-2 Enabling the LSP service



Step 3 Restart the CodeArts Agent CLI to apply the change.

----End

Related Documents

After the local LSP service is enabled, if a message indicating that the LSP service is unavailable is displayed during an agent session, locate and rectify the fault by referring to the operations in **What Should I Do If the LSP Service Is Unavailable?**.

11 Scheduled Tasks

You can create and manage scheduled tasks using natural language, including periodic tasks (such as daily code checks) and one-time tasks (such as tomorrow's submission reminders), achieving flexible task scheduling and control.

Scheduled Task Commands

Table 11-1 Scheduled task commands

Development Environment	Command	Function
TUI	/schedule Natural language prompt	In the TUI development environment, you can create a scheduled task by entering the **/schedule** command or using a natural language description.
CLI	codearts run --command schedule "*Prompt*" codearts run "Prompt"	In the CLI development environment, you can create a scheduled task by entering the **/schedule** command or using a natural language description.

CAUTION

A scheduled task created using a command will be executed immediately, while a scheduled task created using a natural language description will not.

Scheduled Task Configuration Files

Running scheduled tasks in the CodeArts Agent CLI depends on the following files. The data is automatically generated by the system.

Table 11-2 Files on which scheduled tasks depend

File Type	File Path	Description
Task data file (configuration file)	~/.codeartsdoer/cli-data/cron/scheduled_tasks.json	Stores the definitions of all scheduled tasks, such as the interval, prompt, and status. For the specific fields in the data file, see **Table 11-3**.
Execution log file (execution record)	~/.codeartsdoer/cli-data/cron/cron_log/task_*{TASK_ID}*.json	Records the execution result of each task, including the status, duration, and result summary.

Table 11-3 Data file fields

Field	Value Range/Format	Description	Example
id	A random ID consisting of 8 characters (automatically generated by the system)	Unique task ID	10e366d0
cron	A 5-segment Cron expression: minute hour day month weekday Supported formats: *, */N, N, N-M, and N,M,O Value range: minute (0–59), hour (0–23), day (1–31), month (1–12), and weekday (0–6, where **0** indicates Sunday)	Execution schedule (local time), which is used to trigger a task at a scheduled time	*/5 * * * * (every 5 minutes) and 0 9 * * * (9:00 every day)
prompt	Any text	AI prompt (instruction content) executed when a task is triggered	Check the remaining storage space of the local drive C.
projectPath	Absolute path	Project path (The system automatically binds the current project context.)	D:\tmp\test_hc_cli

Field	Value Range/Format	Description	Example
status	• **active**: The task is running. • **paused**: The task is paused. • **expired**: The task has expired. • **deleted**: The task has been deleted.	Current task status (It is automatically managed by the system, and its initial value is **active**.)	active
recurring	• **true**: a recurring task, which is repeatedly executed based on the **cron** interval • **false**: a one-time task, which is executed only once and then automatically ends.	Whether to enable recurring execution	true
durable	• **true**: persistent scheduled task. A persistent task is automatically restored and executed upon restart. A persistent task is created only when you explicitly require cross-restart retention, daily execution, or permanent retention. • **false**: session-level scheduled task. By default, a session-level task is executed. After you exit the program, the task will not be retained, and will not be restored upon the next startup. This task type is suitable for scenarios such as temporary reminders and short-term polling.	Whether the task is a persistent task. The default value is **false**.	false
createdAt	Unix timestamp (ms)	Task creation time, which is automatically generated by the system	1783996637621

Field	Value Range/Format	Description	Example
humanSchedule	Description	Scheduling description, which is automatically generated by the system based on the cron expression	Every minute, 09:00 every day, every 5 minutes
sessionID	String	Session ID. A session is created for each scheduled task. The session is updated with the task.	ses_0a16ceba1ffeXQHnpNW0jh20J8
agentID	Built-in agent name	Agent used to execute a task. It is automatically obtained by the system.	build, code
modelID	ID of a configured model	ID of the model used to execute a task. It is automatically obtained by the system.	deepseek-v3.2
providerID	Provider ID that has been configured	ID of provider of the model used to execute a task. It is automatically obtained by the system.	huaweicloud-maas
isCustom	• **true**: custom agent • **false**: built-in agent	Whether to use a custom agent, which is automatically determined by the system	false
sessionType	**kernel** or other	Session type, which is automatically obtained by the system	kernel
category	• **code** • **office**	Task category, which is automatically obtained by the system	code

Creating a Scheduled Task on the TUI

In the TUI development environment, you can create a scheduled task using a command or natural language. A scheduled task created using a command will be

executed immediately, while a scheduled task created using a natural language description will not.

Creating a Scheduled Task Using the /schedule Command

Command syntax:

```
/schedule [Interval] <Prompt>
```

Table 11-4 Parameters in the /schedule command

Parameter	Description
Interval	(Optional) Interval at which the scheduled task is executed. If this parameter is not set, the command is executed every 10 minutes by default. Format: number + unit letter. The unit letter can be **s** (second), **m** (minute), **h** (hour), or **d** (day).
Prompt	(Mandatory) The command that you want AI to execute

CAUTION

After a scheduled task command is submitted for the first time, AI immediately executes the specified prompt and then sets and runs a scheduled task. Subsequent scheduled tasks will be generated in an independent session. You can run the **/sessions** command to view all sessions. For example, if you deliver a **/schedule 5m check the deployment status** command, AI will immediately check the deployment status once and then check the deployment status every five minutes.

Example:

Step 1 Enter the TUI development environment.

1. Open the root directory of the target project.
2. Right-click in the blank area and select **Open Windows Terminal here**.
3. Enter **/codearts** in the terminal and press **Enter** to enter the TUI development environment.

Step 2 Run the **/schedule** command to create a scheduled task.

1. In the TUI dialog box, enter **/** and select **/schedule**.
2. Enter the following command and press **Enter**:
   ```
   1m Create a TXT file with an increasing suffix and write the current system date and time into the file.
   ```
3. AI creates a scheduled task and executes it immediately. In addition, the two files listed in **Table 11-5** are generated.

Figure 11-1 Creating a session-level scheduled task

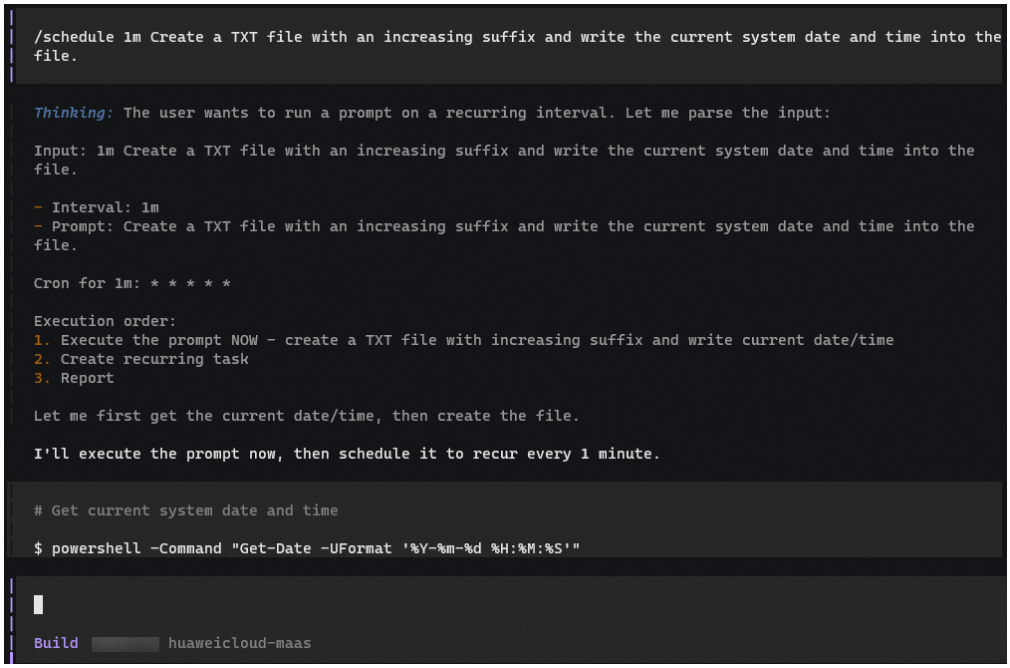


Table 11-5 Generated files

File	Storage Path	Description
scheduled_tasks.json	%USERPROFILE%/.codeartsdoer/cli-data/cron/	Records the current task information.
task_*{TASK_ID}*.json	%USERPROFILE%/.codeartsdoer/cli-data/cron/cron_log/	Records the execution records of the current task.

4. Run the **/sessions** command to view the session created for the scheduled task.

Figure 11-2 Viewing the session created for the scheduled task

----End

Creating a Scheduled Task Using Natural Language

In addition to the **/schedule** command, you can also use a natural language description to create a scheduled task. **Scheduled tasks created using natural language will not be executed immediately.**

Step 1 Enter the TUI development environment.

1. Open the root directory of the target project.
2. Right-click in the blank area and select **Open Windows Terminal here**.
3. Enter **/codearts** in the terminal and press **Enter** to enter the TUI development environment.

Step 2 In the TUI dialog box, enter the operation you want AI to perform and press **Enter**.

- **Example of creating a recurring task using natural language**

  Table 11-6 Creating a recurring task using natural language

Prompt	What AI Will Do
Check error logs in the production environment every 30 minutes.	Create a recurring task that runs every 30 minutes.
Run a unit test at 9:00 a.m. every day.	Create a recurring task that runs at 9:00 every day.
Scan for security vulnerabilities every hour.	Create a recurring task that runs every hour.

- **Example of creating a one-off task using natural language**

  Table 11-7 Creating a one-off task using natural language

Prompt	What AI Will Do
Remind me to review PR#42 at 9:00 a.m. tomorrow.	Create a one-off task that runs at 9:00 tomorrow and automatically ends after execution.
Remind me to confirm the database migration result in 3 minutes.	Create a one-off task that runs once in 3 minutes.
Remind me to submit a weekly report at 10:00 next Monday.	Create a one-off task that runs at 10:00 next Monday.

----End

Creating a Scheduled Task on the CLI

In the CLI development environment, you can create a scheduled task using a command or natural language. A scheduled task created using a command will be executed immediately, while a scheduled task created using a natural language description will not.

CAUTION

CLI dialogs are one-off. The scheduled tasks created on the CLI are only written to the **scheduled_tasks.json** configuration file. Scheduled tasks will be executed only when the TUI or other CodeArts Agent CLI processes are started. Keep CodeArts Agent CLI running to ensure that tasks can be executed stably as scheduled.

Creating a Scheduled Task Using a Command

Command syntax:

```
codearts run --command schedule "Prompt"
```

Table 11-8 Parameters in the command

Parameter	Description
Prompt	(Mandatory) The operation that you want AI to perform

Example:

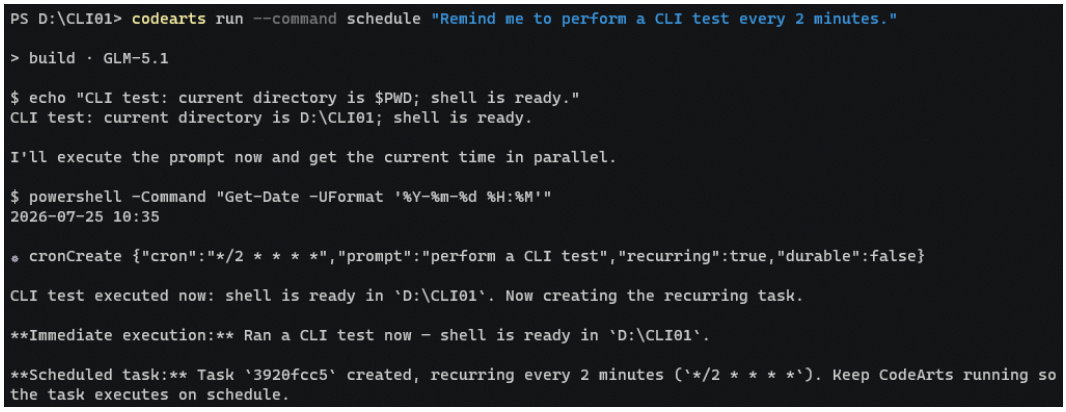
Step 1 In the CLI development environment, enter the command for creating a scheduled task and press **Enter**.

For example, to create a scheduled task that reminds you to perform a CLI test every two minutes, you can use the following prompt. **If no time is specified in the command, a scheduled task that is executed every 10 minutes by default is created** (for example, you can run **codearts run --command schedule** to list all scheduled tasks).

```
codearts run --command schedule "Remind me to perform a CLI test every 2 minutes."
```

Step 2 AI will create a scheduled task based on your instruction and run it immediately.

Figure 11-3 Creating and running a scheduled task



----End

Creating a Scheduled Task Using Natural Language

Step 1 In the CLI development environment, enter the command for creating a scheduled task and press **Enter**.

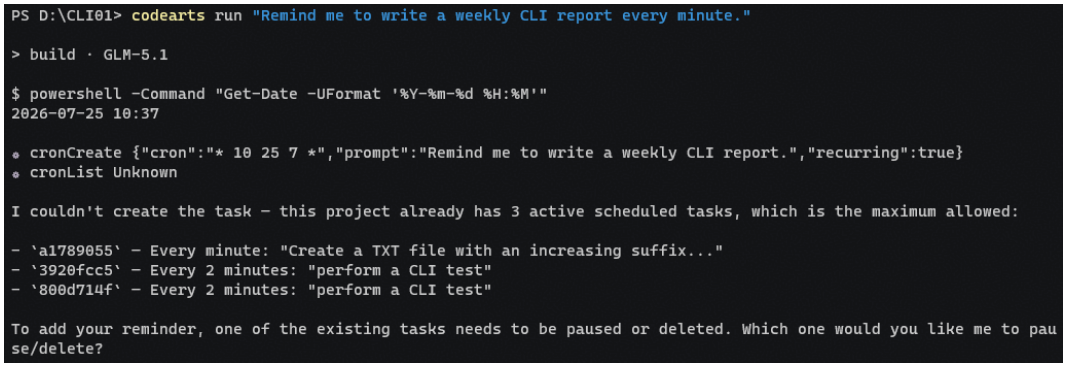
```
codearts run "Prompt"
```

For example, to create a scheduled task that reminds you to write a weekly CLI report every minute, you can use the following prompt.

```
codearts run "Remind me to write a weekly CLI report every minute."
```

Step 2 AI will create a scheduled task based on your instruction.

Figure 11-4 Creating a scheduled task



----End

Managing Scheduled Tasks

Scheduled tasks cannot be managed using commands. You can manage them only using natural language descriptions.

Table 11-9 Managing scheduled tasks

Natural Language Prompt	Execution Result
• TUI: List all scheduled tasks. • CLI: codearts run "List all scheduled tasks."	The list of all scheduled tasks is displayed.
• TUI: View paused scheduled tasks. • CLI: codearts run "List all scheduled tasks."	Only paused tasks are displayed.
• TUI: View details about a task with a specified *task ID*. • CLI: codearts run "View details about a task with a specified *task ID*."	The task details are displayed, including the execution history and next execution time.
• TUI: Pause a task with a specified *ID*. • CLI: codearts run "Pause a task with a specified *ID*."	The task stops, with no configuration lost. After the task is resumed, it continues running at the original interval.
• TUI: Resume a task with a specified *ID*. • CLI: codearts run "Resume a task with a specified *ID*."	The paused task is reactivated and continues running as planned.
• TUI: Delete a task with a specified *ID*. • CLI: codearts run "Resume a task with a specified *ID*."	The task is permanently deleted and cannot be restored.
• TUI: Immediately execute a scheduled task with a specified *ID*. • CLI: codearts run "Immediately execute a scheduled task with a specified *ID*."	The task is immediately executed, without affecting the original schedule. For example, you can immediately execute a scheduled task that is paused.

12 Custom Models

CodeArts Agent CLI supports third-party large language models (LLMs). If you have purchased the CodeArts Agent professional edition, you can also add third-party LLMs on the console. For details, see "Configuring Custom Models".

Constraints

- Only third-party custom enterprise models using the OpenAI-compliant Chat Completions APIs and Anthropic-compliant Messages APIs can be integrated.
- The custom models created on the console can be invoked by all agents. However, those created on the client can be invoked only by the system built-in agents, AgentTeam, and custom agents.
- The provider and model ID combination of each model you add must be unique.

Prerequisites

- Your account has been added as an **enterprise member** and **assigned a seat**.
- If you have purchased CodeArts Agent professional edition, ensure that the enterprise administrator has enabled model customization for members on the console. For details, see **Configuring Custom Models**.

Custom Model Configuration File

CodeArts Agent CLI allows you to manage custom models using a configuration file. On the CLI or TUI, you cannot create, modify, or delete custom models using commands. Instead, you can edit all configurations only by editing the file.

The configuration file for custom models is **codearts_cli.json**, which is stored in **~/.codeartsdoer/**. The tilde (**~**) indicates the home directory of the current user. In Windows, it is equivalent to **C:\Users***Username*\. In macOS, it is equivalent to **/Users/***Username***/**. In Linux, it is equivalent to **/home/***Username***/**.

In the configuration file, **provider** is the top-level key. Each provider ID matches a configuration block. The complete structure of the configuration file is as follows:

```
{
  "provider": {
    "<Provider ID>": {
      "name": "<Provider name>",
```



```
    "npm": "<SDK package name>",
    "api": "<Provider API URL>",
    "env": ["<Environment variable name>", ...],
    "whitelist": ["<Model ID>", ...],
    "blacklist": ["<Model ID>", ...],
    "options": {
      "apiKey": "<API key>",
      "baseURL": "<Base API URL>",
      "enterpriseUrl": "<Enterprise edition URL>",
      "setCacheKey": true,
      "timeout": 300000,
      "chunkTimeout": 60000
    },
    "models": {
      "<Model ID>": {
        "id": "<Model API ID>",
        "name": "<Displayed name>",
        "family": "<Model family>",
        "reasoning": true,
        "attachment": true,
        "temperature": true,
        "tool_call": true,
        "limit": { "context": 128000, "input": 120000, "output": 8000 },
        "cost": {
          "input": 3.0,
          "output": 15.0,
          "cache_read": 0.5,
          "cache_write": 2.0,
          "context_over_200k": {
            "input": 6.0,
            "output": 30.0,
            "cache_read": 1.0,
            "cache_write": 4.0
          }
        },
        "modalities": {
          "input": ["text", "image", "pdf"],
          "output": ["text"]
        },
        "headers": { "X-Custom-Header": "value" },
        "status": "active",
        "provider": { "npm": "<SDK package name>", "api": "<API URL>" },
        "variants": { ... }
      }
    }
  }
}
```

NOTE

apiKey is your core asset. Do not disclose it. After you configure **apiKey** and restart the TUI or CLI, **apiKey** will be automatically encrypted. The ciphertext **apiKey** will then be displayed in the configuration file.

The provider ID is a top-level key in the configuration file and is used to identify the model source. **Table 12-1** lists common provider IDs. You can also use any custom string (for example, **my-server**) as the provider ID, as long as the corresponding model supports OpenAI-compatible API formats.

Table 12-1 Common provider IDs

Provider ID	Description
openai	OpenAI
deepseek	DeepSeek

Provider ID	Description
glm	glm
kimi	kimi

Table 12-2 Parameters in the custom model configuration file

Field	Type	Mandatory	Description
provider.<Provider ID>.name	String	No	Displayed name of the provider. The default value is the provider ID.
provider.<Provider ID>.npm	String	No	SDK package name. The default value is **@ai-sdk/openai-compatible**.
provider.<Provider ID>.api	String	No	API URL of the provider. It is similar to **options.baseURL**.
provider.<Provider ID>.env	String array	No	List of environment variable names. CodeArts Agent CLI reads these environment variables in sequence and uses them to roll back the API key.
provider.<Provider ID>.whitelist	String array	No	Model whitelist. Only models in the list will be loaded. (This list can be used to filter out unnecessary models for built-in providers.)
provider.<Provider ID>.blacklist	String array	No	Model blacklist. Models in the list will not be loaded.
provider.<Provider ID>.options.apiKey	String	Yes	API key. When loaded for the first time, the plaintext key is automatically encrypted in enc:v1: format and written back to the configuration file.

Field	Type	Mandatory	Description
provider.<Provider ID>.options.baseURL	String	Yes	Base URL of the API, for example, **https://your-server.com/v1**
provider.<Provider ID>.options.enterpriseUrl	String	No	GitHub Enterprise URL, which is only used for the authentication of Copilot providers' enterprise editions
provider.<Provider ID>.options.setCacheKey	Boolean	No	Whether to enable promptCacheKey. The default value is **false**.
provider.<Provider ID>.options.timeout	Number or false	No	Request timeout interval, in milliseconds. The default value is **300,000** (5 minutes). If this parameter is set to **false**, timeout is disabled.
provider.<Provider ID>.options.chunkTimeout	Number	No	Timeout interval between blocks in an SSE streaming response, in milliseconds. If no new data block is received within this period, the request is interrupted.
provider.<Provider ID>.options.*	Any type	No	Options support any additional fields (**catchall**). The SDK implementation of the provider may read these custom options.
provider.<Provider ID>.models.<Model ID>.id	String	No	API identifier of the model (model ID sent to the provider). By default, it is the configuration key name.
provider.<Provider ID>.models.<Model ID>.name	String	No	Model name displayed on the CodeArts Agent CLI client. By default, it is the configuration key name.

Field	Type	Mandatory	Description
provider.<Provider ID>.models.<Model ID>.family	String	No	Model family name (such as **gpt** and **claude**), which is used for grouping models
provider.<Provider ID>.models.<Model ID>.reasoning	Boolean	No	Whether reasoning and thinking capabilities are supported
provider.<Provider ID>.models.<Model ID>.attachment	Boolean	No	Whether attachments such as images and PDF files can be uploaded
provider.<Provider ID>.models.<Model ID>.temperature	Boolean	No	Whether the temperature can be adjusted
provider.<Provider ID>.models.<Model ID>.tool_call	Boolean	No	Whether tools or functions can be called. The default value is **true**.
provider.<Provider ID>.models.<Model ID>.limit.context	Number	No	Maximum context window size (token)
provider.<Provider ID>.models.<Model ID>.limit.input	Number	No	Maximum number of input tokens
provider.<Provider ID>.models.<Model ID>.limit.output	Number	No	Maximum number of output tokens
provider.<Provider ID>.models.<Model ID>.cost.input	Number	No	Unit price of input tokens (price per million tokens)
provider.<Provider ID>.models.<Model ID>.cost.output	Number	No	Unit price of output tokens
provider.<Provider ID>.models.<Model ID>.cost.cache_read	Number	No	Unit price of cache reading
provider.<Provider ID>.models.<Model ID>.cost.cache_write	Number	No	Unit price of cache writing

Field	Type	Mandatory	Description
provider.<Provider ID>.models.<Model ID>.cost.context_over_200k	Object	No	Alternative prices when the context has over 200K tokens, including **input**, **output**, **cache_read**, and **cache_write**
provider.<Provider ID>.models.<Model ID>.modalities.input	String array	No	Supported input modalities, including **text**, **audio**, **image**, **video**, and **pdf**
provider.<Provider ID>.models.<Model ID>.modalities.output	String array	No	Supported output modalities, including **text**, **audio**, **image**, **video**, and **pdf**
provider.<Provider ID>.models.<Model ID>.headers	String of key-value objects	No	Custom HTTP request header, which is sent with the model request
provider.<Provider ID>.models.<Model ID>.status	String	No	Model status, which can be **active**, **beta**, **alpha**, or **deprecated**
provider.<Provider ID>.models.<Model ID>.provider.npm	String	No	Name of the SDK package used by the model, which overwrites the default value from the provider
provider.<Provider ID>.models.<Model ID>.provider.api	String	No	API URL of the model, which overwrites the default value from the provider
provider.<Provider ID>.models.<Model ID>.variants	Key-value object	No	Variants

Custom Model Configuration Example

The following steps describe how to customize a model with the minimum configuration in Windows by setting the **provider ID**, **API key**, and **model**.

Step 1 Go to the **%USERPROFILE%/.codeartsdoer** directory and find and open the **codearts_cli.json** configuration file.

Step 2 Add the following provider information to the **codearts_cli.json** file:

```
{
  "$schema": "https://opencode.ai/config.json",
  "provider": {
    "deepseek": {
```



```
    "options": {
      "apiKey": "sk-***",
      "baseURL": "https://api.deepseek.com/v1"
    },
    "models": {
      "deepseek-chat": {
        "name": "DeepSeek Chat"
      },
      "deepseek-reasoner": {
        "name": "DeepSeek Reasoner",
        "reasoning": true
      }
    }
  }
 },
 "mcp": {},
 "lsp": false
}
```

You need to purchase an API key from the DeepSeek official website and then set **apiKey**.

Step 3 Save the file and exit.

Step 4 Check whether the custom model has been added.

- Run the following command in the CLI environment and check whether the custom model is included in the model list displayed in the command output:
  ```
  codearts models
  ```
- In the TUI environment, run the following slash command to go to the **Select model** page and check whether the custom model is available:
  ```
  /models
  ```

----End

Specifying the Model to Run

In CodeArts Agent CLI, you can specify a model to run your task. In the following command, *provider* indicates the model provider, and *model* indicates the model name.

- TUI development environment
  - When **starting** TUI, you can run the following command to specify the model to run:
    ```
    codearts --model provider/model
    You can also use a short parameter.
    codearts -m provider/model
    ```
  - In the TUI development environment, you can enter the **/models** command to quickly switch models.
- CLI development environment
  ```
  codearts run --model provider/model "Your question"
  ```


13 MCP

The Model Context Protocol (MCP) server of CodeArts Agent CLI can run in either of the following modes:

- Local mode: CodeArts Agent CLI starts and communicates with the processes running on your computer.
- Remote mode: CodeArts Agent CLI communicates with the processes running on other machines or servers through HTTP or WS.

After configuring an MCP file, you can run the **/mcps** command on the TUI to view the file.

13.1 Default MCP Server Configuration

You can configure a project-level or user-level MCP server. **Table 1 MCP server configuration paths** lists the configuration paths.

Table 13-1 MCP server configuration paths

Level	Application Scope	Configuration Path
Project	Current project Priority: project-level > user-level	*Project root directory*/**.codeartsdoer/codearts_cli.jsonc** or *Project root directory*/**.codeartsdoer/codearts_cli.json**
User	All projects of the current user	**~/.codeartsdoer/codearts_cli.jsonc** or **~/.codeartsdoer/codearts_cli.json** The tilde (~) indicates the home directory of the current user. In Windows, it is equivalent to **C:\Users***Username*****. In macOS, it is equivalent to **/Users/***Username***/**. In Linux, it is equivalent to **/home/***Username***/**.

Local MCP Server

You can configure a local MCP server.

Parameters

Table 13-2 Local MCP server parameters

Parameter	Type	Description
type	String	This parameter is mandatory. The value must be **local**.
command	String array	Command and input parameters for starting the MCP service. This parameter is mandatory.
environment	Object	Environment variables set for running the server
enabled	Boolean	Whether to enable the MCP server
timeout	Number	Response timeout interval for obtaining the MCP service tool, in milliseconds. The default interval is 100,000 ms.

Configuration Example

This section uses Windows as an example to describe how to configure a local MCP server in a personal environment.

Step 1 In the **C:/Users/***Username***/.codeartsdoer/** directory, create the **codearts_cli.jsonc** file, **write the following configuration to the file, and restart the CodeArts Agent CLI**:

```
{
  "mcp": {
    // Custom MCP service name: character analysis tool
    "mcp-character-tools" : {
      // Mandatory. Startup command: npx is used to start the local MCP service.
      "command" : [ "npx", "mcp-character-tools" ],
      // Mandatory. Service type: local MCP server
      "type": "local"
    }
  }
}
```

Step 2 Enter the TUI development environment.

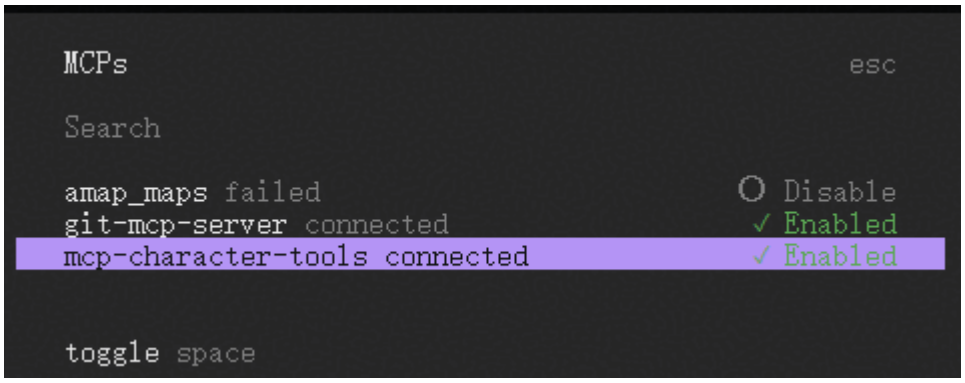
1. Open the root directory of the target project.
2. Right-click in the blank area and select **Open Windows Terminal here**.
3. Enter **/codearts** in the terminal and press **Enter** to enter the TUI development environment.

Step 3 Run the following command to check whether the configuration has taken effect:

```
/mcps
```

The status of **mcp-character-tools** is **Enabled**, indicating that the MCP server is enabled.

Figure 13-1 Status



Step 4 In the TUI dialog box, enter the following instruction.

```
How do I use mcp-character-tools?
```

The function description and usage method are returned, indicating that the configuration of the local MCP server has taken effect.

----End

Remote MCP Server

You can configure a remote MCP server.

Parameters

Table 13-3 Remote MCP server parameters

Parameter	Type	Description
type	String	This parameter is mandatory. The value must be **remote**.
url	String	(Mandatory) URL of the remote MCP server
headers	Object	Request header
environment	Object	Environment variables set for running the server
oauth	Object	OAuth authentication configuration
enabled	Boolean	Whether to enable the MCP server
timeout	Number	Response timeout interval for obtaining the MCP service tool, in milliseconds. The default interval is 5,000 ms.

Configuration Example

This section uses Windows as an example to describe how to configure a remote MCP server in a personal environment.

Step 1 In the **C:/Users/***Username***/.codeartsdoer/** directory, create the **codearts_cli.jsonc** file, **write the following configuration to the file, and restart the CodeArts Agent CLI**:

```
{
  "mcp": {
    // EdgeOne Pages Deploy MCP is a dedicated service that allows you to quickly deploy web applications on EdgeOne Pages and generate public access links. This enables you to instantly preview and share AI-generated web pages.
    "edgeone-pages": {
      "type": "remote",
      "url": "https://mcp-on-edge.edgeone.site/mcp-server",
      "enabled": true,
      "timeout": 45000,
      "headers": {
        // API request authentication header, which is optional
        "Authorization": "Bearer edgeone_token_your_secret"
      }
    }
  }
}
```

Step 2 Enter the TUI development environment.

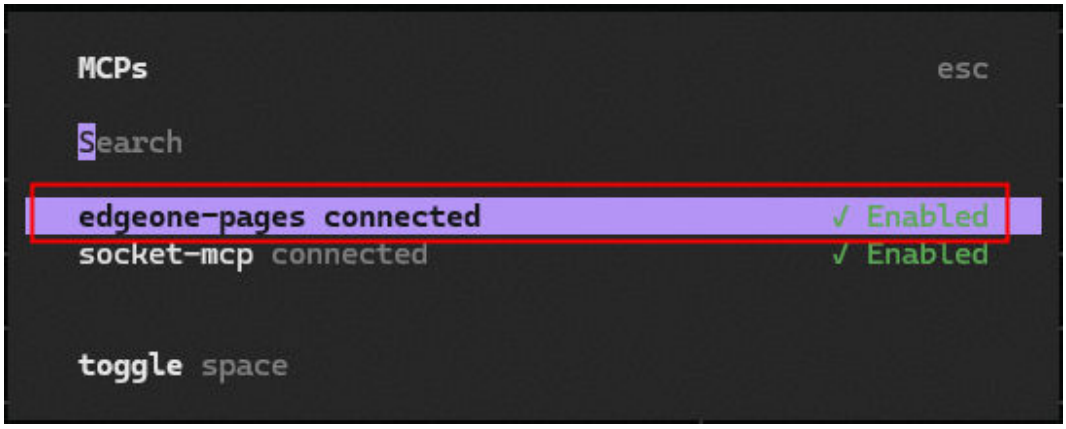
1. Open the root directory of the target project.
2. Right-click in the blank area and select **Open Windows Terminal here**.
3. Enter **/codearts** in the terminal and press **Enter** to enter the TUI development environment.

Step 3 Run the following command to check whether the configuration has taken effect:

```
/mcps
```

The status of **edgeone-pages** is **Enabled**, indicating that the MCP server is enabled.

Figure 13-2 Status

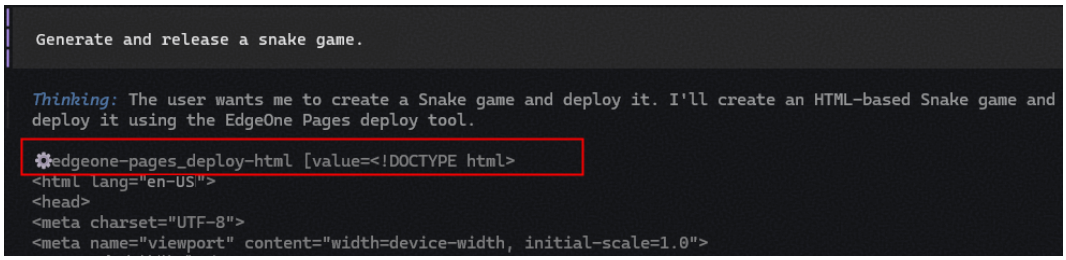


Step 4 In TUI development mode, enter the following instruction and press **Enter**:

```
Generate and release a snake game.
```

AI invokes the MCP server and creates a snake game.

Figure 13-3 MCP server invoked successfully



----End

13.2 Claude Code

In addition to the native configuration, CodeArts Agent CLI supports Claude Code for MCP server configuration. If you already have the MCP configuration file of Claude Code, you can use it in CodeArts Agent CLI without rewriting it. The two methods are equivalent in functions. You can choose either of them.

You can configure a project-level MCP server. **Table 1 MCP server configuration path** lists the configuration path.

Table 13-4 MCP server configuration path

Level	Application Scope	Configuration Path
Project	Current project	*Project root directory*/**.codeartsdoer/mcp/mcp_settings.json**

Local MCP Server

The stdio transmission mode is used to connect to the MCP server through a local process.

Parameters

Table 13-5 Local MCP server parameters

Parameter	Type	Description
command	String	Command for starting the MCP server process, such as **npx**, **uvx**, **node**, and **python**. This parameter is mandatory.
args	String array	Parameters of the startup command

Parameter	Type	Description
env	Object	Environment variables, which are used to transfer configurations such as the API key
disabled	Boolean	Whether to enable the MCP server
timeout	Numeric	Connection timeout interval, in milliseconds

Configuration Example

This section uses Windows as an example to describe how to configure a local MCP server.

Step 1 In the **D:/tmp/test_hc_cli/.codeartsdoer/** directory, create the **mcp/mcp_settings.json** file, **write the following configuration to the file, and restart the CodeArts Agent CLI**:

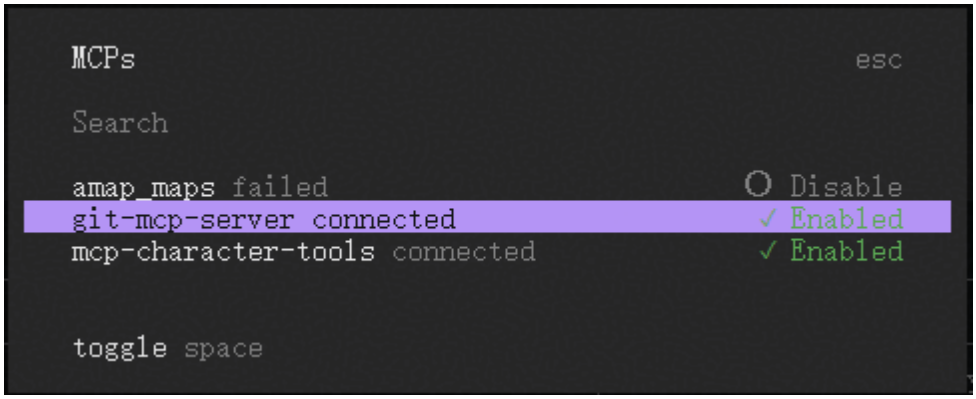
```
{
  "mcpServers": {
     "git-mcp-server": {
     "transportType": "stdio",
     "disabled": false,
     "timeout": 100000,
     "args": [
       "@cyanheads/git-mcp-server"
     ],
     "command": "npx",
     "env": {
       "GIT_SIGN_COMMITS": "false",
       "MCP_LOG_LEVEL": "info"
     }
    }
  }
}
```

Step 2 Run the following command to check whether the configuration has taken effect:

```
/mcps
```

The status of **git-mcp-server** is **Enabled**, indicating that the MCP server is enabled.

Figure 13-4 Status



Step 3 In the TUI dialog box, enter the following instruction.

```
What can git-mcp-server do?
```

The function description and usage method are returned, indicating that the configuration of the local MCP server has taken effect.

----End

Remote MCP Server

Streamable HTTP is used to connect to the remote MCP server.

Parameters

Table 13-6 Remote MCP server parameters

Parameter	Type	Description
url	String	HTTP URL of the MCP server. This parameter is mandatory.
headers	Object	HTTP request header, which is used for authentication
disabled	Boolean	Whether to enable the MCP server
timeout	Numeric	Connection timeout interval, in milliseconds
autoApprove	String array	List of tools that are automatically approved

Configuration Example

This section uses Windows as an example.

Step 1 In the **D:/tmp/test_hc_cli/.codeartsdoer/** directory, create the **mcp/mcp_settings.json** file, **write the following configuration to the file, and restart the CodeArts Agent CLI**:

```
{
  "mcpServers": {
    "socket-mcp": {
      "url": "https://mcp.socket.dev/",
      "headers": {
        "Content-Type": "application/json"
      },
      "timeout": 30000,
      "disabled": false
    }
  }
}
```

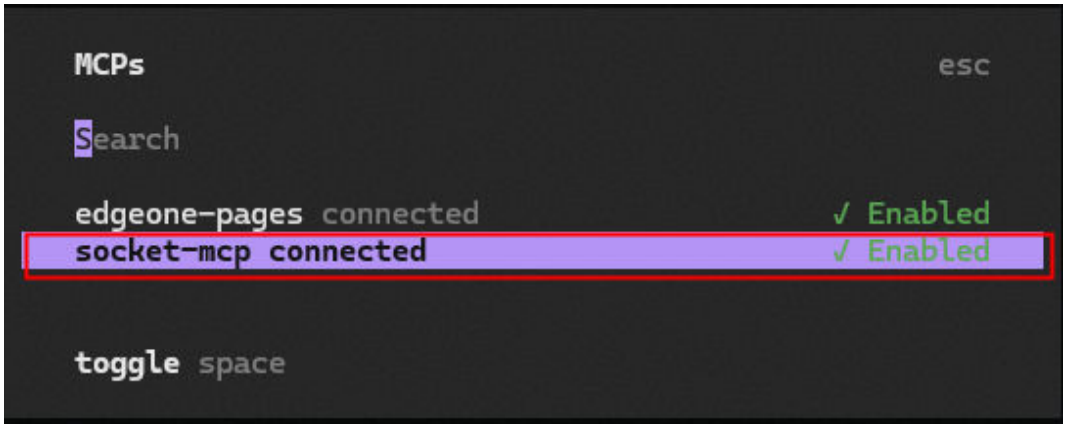
The MCP server scans dependencies to evaluate the quality and security of software packages.

Step 2 Run the following command to check whether the configuration has taken effect:

```
/mcps
```


The status of **socket-mcp** is **Enabled**, indicating that the MCP server is enabled.

Figure 13-5 Status

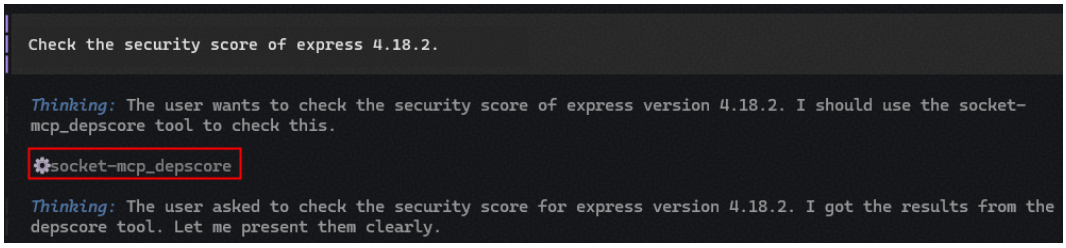


Step 3 In the TUI dialog box, enter the following command and press **Enter**:

```
Check the security score of express 4.18.2.
```

The security score is returned, indicating that the configuration of the remote MCP server has taken effect.

Figure 13-6 MCP server configuration has taken effect



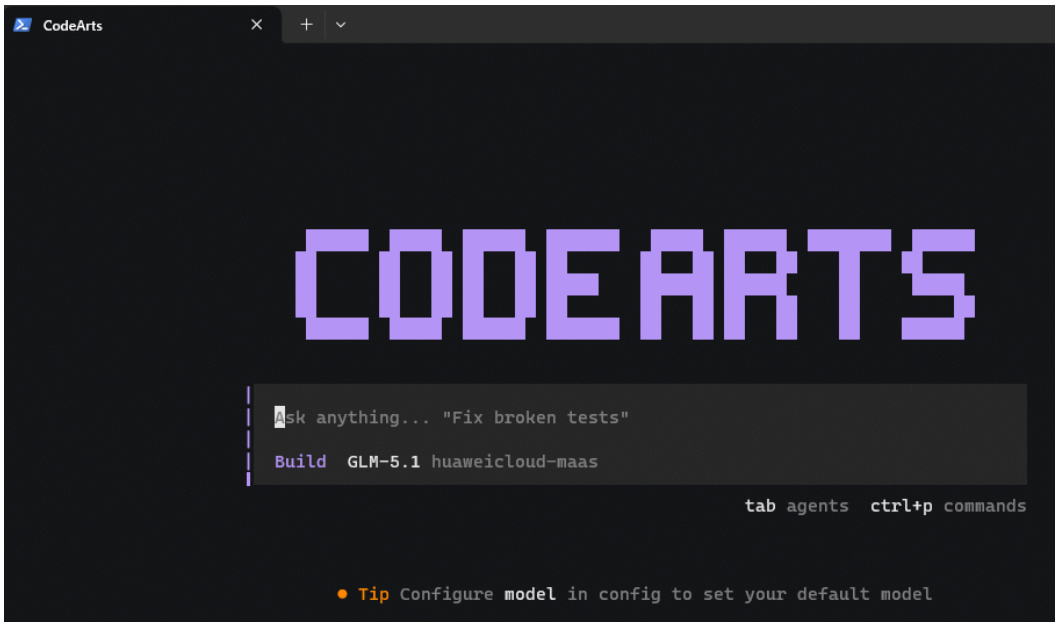
----End

14 TUI

14.1 Slash Commands

You can enter **codearts** in the CMD, PowerShell, or terminal and press **Enter** to enter the TUI development mode.

Figure 14-1 TUI development mode



In TUI development mode, you can enter a slash (/) followed by a command name to quickly perform an operation.

Table 14-1 Slash commands supported by TUI

Command	Description
/help	Displays the help dialog box. **NOTE** After the dialog box is displayed, click **ok**, **esc**, or **Enter** to use shortcut keys to view shortcut operation prompts.
/mcps	Displays the list of MCPs. You can press the space key to enable or disable an MCP. **NOTE** If an MCP is disabled, it cannot be used by LLMs.
/compact	Compresses the current session.
/exit	Exits the TUI and returns to the PowerShell or CMD interface.
/q	
/quit	
/logout	
/init	Scans a project, learns its purpose, and generates an AGENTS.md file. **NOTE** If your project does not have source code, configuration, or build files, TUI will advise you to create an AGENTS.md file in a project with code or on your working disk.
/models	Checks and switches models.
/new	Starts a new session.
/undo	Cancels the last message in the session. You can edit the task again or cancel it directly. **CAUTION** Any file changes will also be undone.
/redo	Re-executes the message that was canceled. This command can be used only after **/undo** is executed. **CAUTION** All file changes will also be restored.
/sessions	Lists sessions and switches between them. After selecting a session, you can perform the following operations: ● Press **Ctrl+D** to delete the session. After confirmation, the session will be deleted. ● Press **Ctrl+R** to rename the session. After changing the session name, press **Enter** to apply the change.
/resume	
/continue	
/themes	Lists available themes.

Command	Description
/thinking	Whether to display the dialog inference process. It is enabled by default, which means you can view the inference details of the models with deep thinking capabilities. **NOTE** This command only controls whether to display the thinking block and does not enable or disable the inference capability of models.
/privacy	Views the privacy policy.
/docs	Views the documentation.
/lsp-toggle	Enables or disables the LSP service, or views the LSP service status.
/sdd-new	Creates SDD requirement specifications.
/sdd-design	Creates an SDD technical design.
/sdd-tasks	Creates an SDD encoding task.
/sdd-apply	Applies SDD encoding.
/run-mode	Switches or views the sandbox run mode.
/bugfix	Automatically invokes the repair expert agent to fix code issues based on the descriptions in the trouble tickets submitted by users.
/schedule	Creates a scheduled task. For details, see **Creating a Scheduled Task on the TUI**.
/agents	Checks and switches agents.
/codebase/init	Creates repository indexes. Run this command in the project directory to create repository indexes.
/codebase/detail	Queries the repository indexing progress.
/codebase/update	Incrementally updates repository indexes.
/codebase/delete	Deletes repository indexes.
/editor	Opens the text editor.
/review	Reviews code changes.
/skills	Queries skills.
/status	Checks the status.

14.2 Shortcut Keys

The previous section lists some shortcut keys corresponding to slash commands. The TUI development environment also supports built-in shortcut keys that you can use to operate the input information in the dialog box.

Table 14-2 Shortcut keys for operating prompts in the TUI dialog box

Shortcut Key	Operation
Ctrl+A	Moves the cursor to the beginning of the current line.
Ctrl+E	Moves the cursor to the end of the current line.
Ctrl+B	Moves the cursor one character to the left.
Ctrl+F	Moves the cursor one character to the right.
Windows/Linux: Alt +B macOS: Option+B	Moves the cursor one word to the left.
Windows/Linux: Alt+F macOS: Option+F	Moves the cursor one word to the right.
Ctrl+D	Deletes the character at the cursor position.
Ctrl+K	Deletes the content from the cursor position to the end of the line.
Ctrl+U	Deletes the content from the beginning of the line to the cursor position.
Ctrl+W	Deletes the previous word.
Windows/Linux: Alt +D macOS: Option+D	Deletes the next word.
Ctrl+G	Goes to the first turn of the session.

14.3 SDD

Specification-driven development (SDD) contains four phases: determination of required specifications, technical design, task planning, and code implementation. This structured process ensures that each step is controllable and that the history is traceable.

SDD Commands

Table 14-3 SDD commands

Command	Input	Output Path	Function
/sdd-new	/sdd-new + *Requirement description* (mandatory)	*{Project root path}*/.codeartsdoer/specs/*Folder name of the generated file*/spec.md	Generates the **spec.md** file based on the requirement description to define **what to do**, such as business behavior and requirement constraints.
/sdd-design	/sdd-design + *Folder name of the generated file* (optional). The system infers the folder name of the file generated in the previous phase from the dialog context. **CAUTION** The directory (*{Project root path}*/.codeartsdoer/specs/*Folder name of the generated file*) and the generated **spec.md** file are mandatory.	*{Project root path}*/.codeartsdoer/specs/*Folder name of the generated file*/design.md	Generates the **design.md** file based on the **spec.md** file. The **design.md** file defines **how to do it**, such as the technical architecture and implementation solution.

Command	Input	Output Path	Function
/sdd-tasks	/sdd-tasks + *Folder name of the generated file* (optional). The system infers the folder name of the file generated in the previous phase from the dialog context. **CAUTION** **spec.md** and **design.md** are mandatory.	*{Project root path}*/.codeartsdoer/specs/*Folder name of the generated file*/tasks.md	Generates the task list **tasks.md** based on **spec.md** and **design.md**.
/sdd-apply	/sdd-apply + *Folder name of the generated file* (optional). The system infers the folder name of the file generated in the previous phase based on the dialog context. **This command is executed in the primary agent.** **CAUTION** **spec.md**, **design.md**, and **tasks.md** are mandatory.	Code and updated tasks	Executes coding tasks one by one based on **tasks.md**.

NOTE

The *Folder name of the generated file* is automatically generated by a model. It may vary even if the same requirement description is input.

Generating Code Using SDD

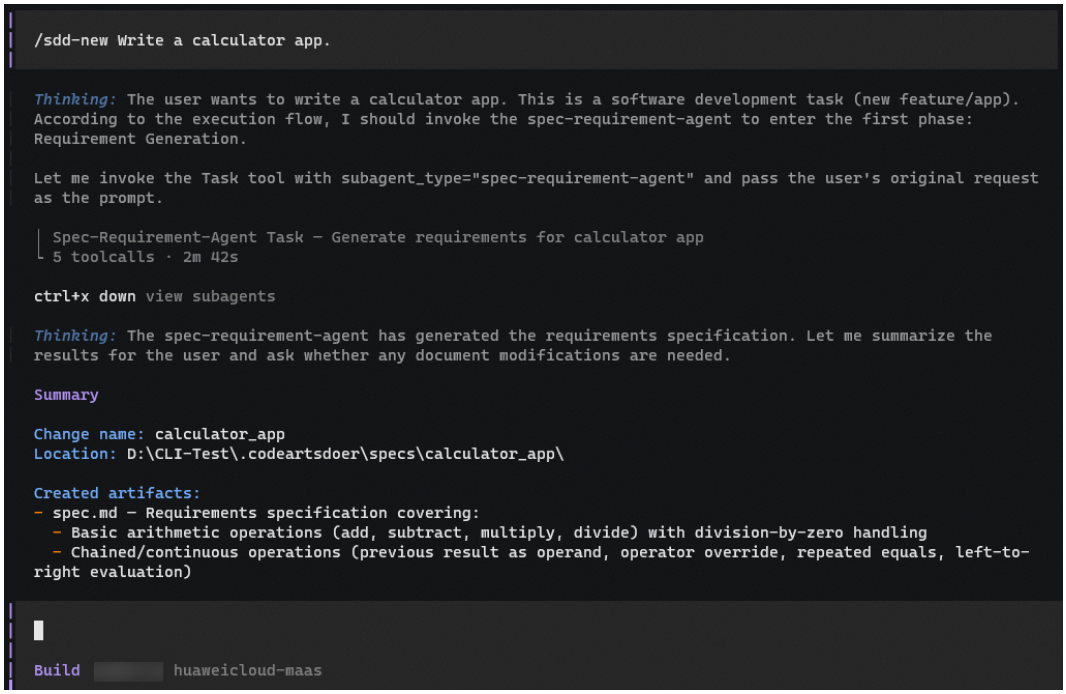
Step 1 Enter the TUI development environment.

1. Open the root directory of the target project.

2. Right-click in the blank area and select **Open Windows Terminal here**.
3. Enter **/codearts** in the terminal and press **Enter** to enter the TUI development environment.

Step 2 **Create requirement specifications.** In the TUI dialog box, enter **/sdd-new** *Requirement description* and press **Enter** to generate the **spec.md** file.

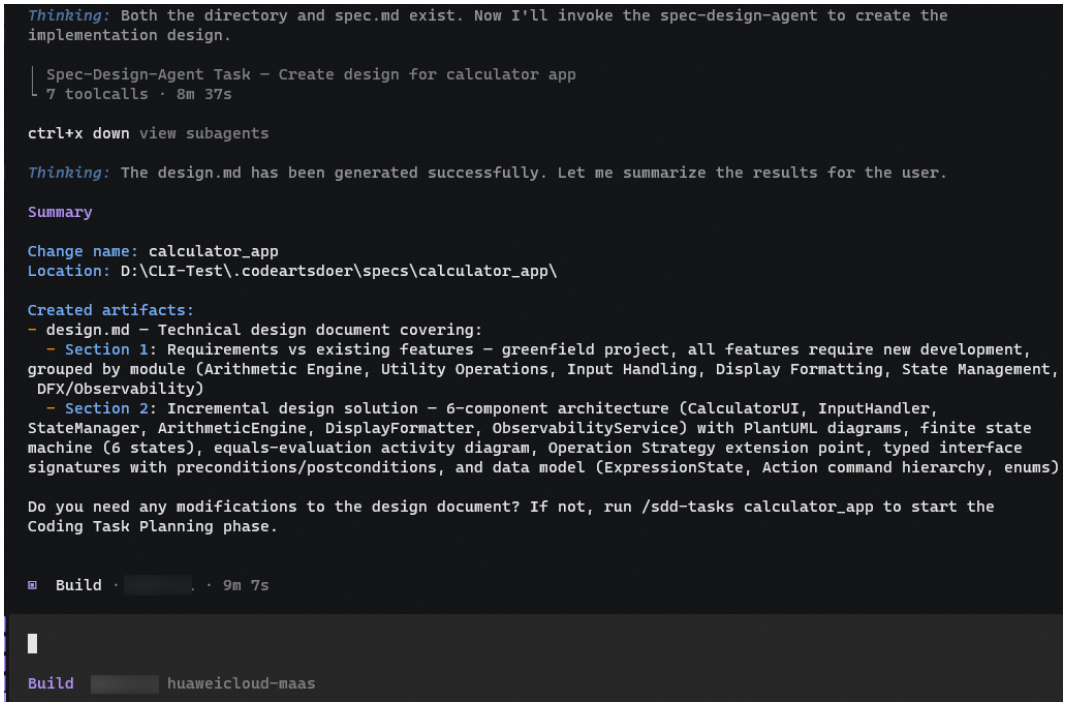
Figure 14-2 Creating requirement specifications



Open *{Project root path}*/**.codeartsdoer/specs/***Name of the generated file*. You can find the generated **spec.md** file. If the generated **spec.md** file does not meet your expectations, you can manually modify it or use the agent to modify it.

Step 3 **Create a technical design.** In the TUI dialog box, enter **/sdd-design** and press **Enter** to generate the **design.md** file based on the **spec.md** file.

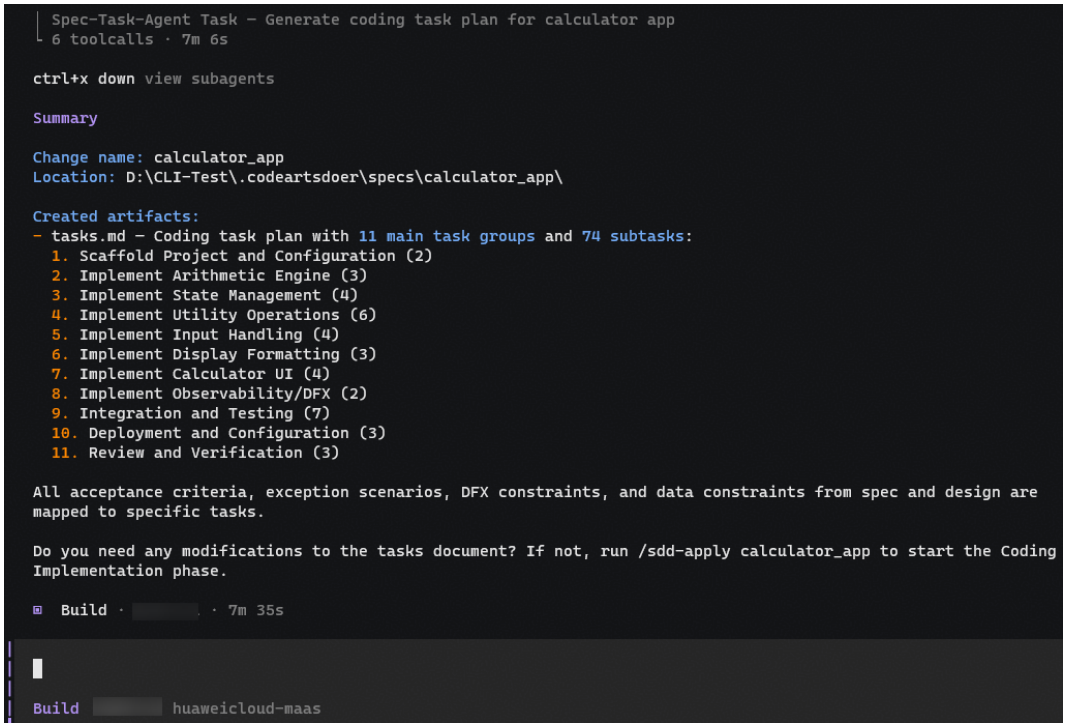
Figure 14-3 Creating a technical design



Open *{Project root path}*/**.codeartsdoer/specs/***Name of the generated file*. You can find the generated **design.md** file. If the generated **design.md** file does not meet your expectations, you can manually modify it or use the agent to modify it.

Step 4 **Create a coding task list.** In the TUI dialog box, enter **/sdd-tasks** and press **Enter** to generate the **tasks.md** file based on the **spec.md** and **design.md** files.

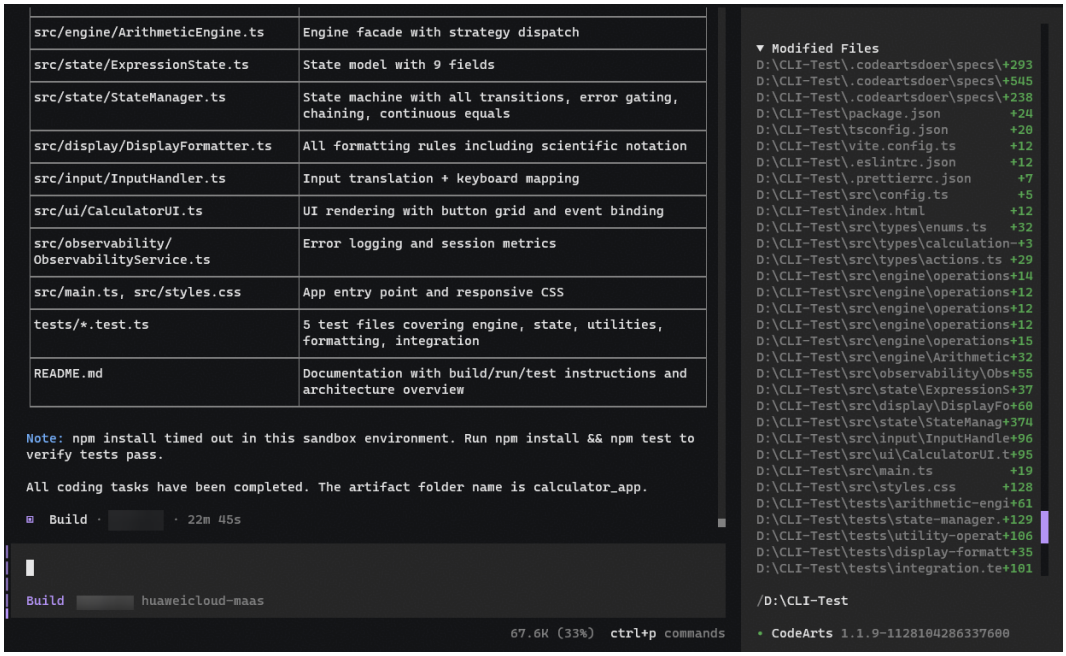
Figure 14-4 Generating a coding task list



Open *{Project root path}*/**.codeartsdoer/specs/***Name of the generated file*. You can find the generated **tasks.md** file. If the generated **tasks.md** file does not meet your expectations, you can manually modify it or use the agent to modify it.

Step 5 **Perform coding tasks.** In the TUI dialog box, enter **/sdd-apply** and press **Enter**. The agent will execute the coding tasks one by one based on the **tasks.md** file.

Figure 14-5 Performing coding tasks



----End

15 CLI

15.1 Commands

Open the CMD, PowerShell, or terminal and run **codearts [command]** to enter the CLI development mode of CodeArts Agent.

Table 15-1 CLI commands

Command	Description	Subcommand or Subparameter
codearts help	Views command help information.	None
codearts completion	Generates a shell autocomplete script.	None
codearts mcp	Manages the MCP server.	For details about subcommands, see **MCP Commands**.
codearts [project]	Starts the TUI development environment of a specified project. [project] is an optional parameter indicating the project name. For example, you can run the following command to enter the TUI development mode and load the **Test** project: `codearts D:\Test`	None

Command	Description	Subcommand or Subparameter
codearts run [message]	Runs CodeArts Agent CLI in non-interactive mode and directly inputs a prompt. For example, run the following command to read files in the current directory: `codearts run read file`	For details about the parameters supported by the **codearts run** command, see **Table 15-2**.
codearts agent	Manages agents.	• **create**: creates an agent. • **list**: lists all available agents.

codearts models [provider]	Lists available models.
	• For example, you can run the following command to obtain available models of all vendors:
	`codearts models`
	• For example, you can run the following command to obtain available models of the **huaweicloud-maas** vendor:
	`codearts models huaweicloud-maas`
		None

codearts stats	Queries token usage and cost statistics.	Run the following command to query all statistics: `codearts stats` For details about other subcommands, see **Table 15-3**.
codearts session	Manages sessions.	• **list**: lists all sessions. • **delete <sessionID>**: **sessionID** is mandatory. You can obtain it by running the **codearts session list** command. After running the command, select the session to be deleted and press **Enter**.

Command	Description	Subcommand or Subparameter

codearts export [sessionID]	Prints the data of the session with a specified ID in JSON format. You can obtain the **sessionID** by running the **codearts session list** command.
	• For example, you can run the following command, select the session to be exported, and press **Enter** to print the session data.
	`codearts export`
	• For example, you can run the following command to print the data of the session with a specified ID:
	`codearts export [sessionID]`
		None

codearts import <file>	Imports session data from a local JSON file. The **<file>** parameter is mandatory. For example, you can run the following command to import data from a specified file. The imported data includes the **sessionID**. `codearts import session.json` Application scenario: You can continue the session based on the generated **sessionID**.	None
codearts upgrade	Upgrades CodeArts Agent CLI to the latest version.	None
codearts uninstall	Uninstalls CodeArts Agent CLI and deletes all related files.	None
codearts docs	Obtains the help documentation URL of CodeArts Agent CLI.	None
codearts privacy	Queries the privacy policy of CodeArts Agent CLI.	None
codearts debug	Debugs issues.	For details about subcommands, see **Table 15-7**.

Command	Description	Subcommand or Subparameter
codearts --version	Prints the version.	None
codearts --print-logs	Prints logs to stderr.	None
codearts --log-level <Log level>	Sets the log level, which can be **DEBUG**, **INFO**, **WARN**, or **ERROR**.	None
codearts codebase	Manages repository indexes.	For the parameters supported by this command, see **Table 9-3**.

codearts serve	Starts a server without a GUI.
	Before running this command, run the following command in PowerShell, CMD, or the terminal to configure the username and password for the server. Otherwise, error message "Error: CODEARTS_SERVER_PASSWORD is not set; please configure the password to start the server." is displayed.
	`export CODEARTS_SERVER_USERNAME="Username for logging in to the server"`
	`export CODEARTS_SERVER_PASSWORD="Password for logging in to the server"`
	You need to set the username and password by yourself.
		For the parameters supported by this command, see **Table 15-10**.

codearts attach <url>	Binds a local command line to a remote CodeArts Agent CLI instance using a username and a password. You can pass the username and password using environment variables or command line--password.	For the parameters supported by this command, see **Table 15-11**.

Table 15-2 Parameters supported by codearts run

Flag	Description

codearts run --command	• Runs a command using **--command** <*Command*>.
	For example, you can run the following command to perform initialization:
	`codearts run --command init`
	For example, you can run the following command to run the **test** command:
	`codearts run --command test`
	• Generates code using an SDD command in **--command** <*SDD command*> <*Parameter*> format.
	The *SDD command* can be **sdd-new**, **sdd-design**, **sdd-tasks**, or **sdd-apply**. *Parameter* is mandatory.
	– When the command is **sdd-new**, the parameter indicates the description of the task to be executed.
	– When the command is **sdd-design**, **sdd-tasks**, or **sdd-apply**, the parameter indicates the folder name of the product.
	When using SDD commands in the CLI, you do not need to add a slash (/) before them. Other functions are the same as those on the TUI. For details, see **SDD**.
	• Creates a scheduled task using the --command schedule "Prompt" command. For details, see **Creating a Scheduled Task on the CLI**.

codearts run --continue	Continues the previous session. It can be abbreviated as **-c**. For example, you can run the following command to continue the previous session: `codearts run --continue`
codearts run "Command to be executed" --sessionID <sessionID>	Continues a session with a specified ID. It can be abbreviated as **-s**. For example, you can run the following command to continue the session with the specified ID: `codearts run "What day is it today?" --sessionid ses_07885a8e7ffe8ESvH4zV1BdFJw` In the preceding command, **ses_07885a8e7ffe8ESvH4zV1BdFJw** is the session ID. You can obtain it by running the **codearts session list** command.
codearts run --fork	Continues a fork session.
codearts run --model	Model to be used. It can be abbreviated as **-m**.
codearts run --agent	Agent to be used

Flag	Description
codearts run --file	File to be attached to the message. It can be abbreviated as **-f**.
codearts run --format	Format, which can be **default** or **json**
codearts run --title	Session title
codearts run --thinking	Displays thinking blocks.
codearts run --attach	Connects to a running server. For example, you can run the following command to connect to a running CodeArts server whose URL is http://localhost:4096 and perform Q&A: `codearts run --attach http://localhost:4096 --password 1***5 "hi"`
codearts run --auto	Runs the Bash tool in automatic mode in the current session.
codearts run --sandbox	Runs the Bash tool in sandbox mode in the current session.

Table 15-3 Parameters supported by codearts stats

Command	Description

codearts stats --days	Displays statistics of the last *N* days.
	• For example, you can run the following command to display statistics of all days:
	`codearts stats --days`
	• Run the following command to display statistics of the last two days:
	`codearts stats --days 2`

codearts stats --tools	Displays statistics of tools.
codearts stats --models	Displays usage details of all models.
codearts stats --project	Displays statistics of a project. Go to the root directory of the **Git repository** to be queried, right-click in the blank area, and select **Open Windows Terminal here**. Run the following command to query the statistics of the current project: `codearts stats --project` **NOTE** If the target repository has no Git commit records, the system displays project data of all non-Git repositories.

Table 15-4 Flags supported by codearts [project]

Flag	Abbreviation	Description
--continue	-c	Enters the TUI development mode and continues the previous session. For example, you can run the following command to enter the **D:\Test** project and the TUI development mode to continue the session: `codearts D:\Test -c`
--session	-s	ID of the session to be continued. You can obtain the **sessionID** by running the **codearts session list** command. `codearts D:\Test -s <sessionID>`
--fork	-	Creates a fork session based on a specified session. The subcommands are as follows: • **codearts --fork -c**: enters the TUI development mode and continues the previous session. • **codearts --fork -s sessionID**: creates a fork session based on a specified session ID.
--model	-m	Model to be used
--agent	-	Agent to be used

Table 15-5 Parameters supported by codearts session list

Flag	Abbreviation	Description
--max-count	-n	Maximum number of recent sessions
--format	-	Output format, which can be **table** or **json**

Table 15-6 Flags of the codearts uninstall command

Flag	Abbreviation	Description
--keep-config	-c	Retains the configuration file.
--keep-data	-d	Retains session data.
--dry-run	-	Displays the content to be deleted.
--force	-f	Skips the confirmation prompt.

Table 15-7 codearts debug subcommands

Commands	Description
codearts debug config	Definition: displays the complete configuration after parsing. Function: This command displays all configurations that have taken effect (including default values and environment variables) for troubleshooting configuration loading issues.
codearts debug rg	Definition: debugs the ripgrep code search tool. For details about subcommands, see **Table 15-8**. Function: This command is used to debug ripgrep-related behavior, such as searching for rules, ignoring files, and matching paths.
codearts debug file	Definition: debugs file systems. For details about subcommands, see **Table 15-9**. Function: This command is used to debug file-related issues, such as file reading/writing, path parsing, permissions, and symbolic links.
codearts debug scrap	Definition: lists all known projects. Function: This command helps developers learn the projects identified by CodeArts Agent CLI.
codearts debug skill	Definition: lists all available skills. Function: This command returns all the skills loaded in the current environment, including their names, descriptions, locations, and content.
codearts debug agent <name>	Definition: queries the configuration details of a specified agent. <name> is a mandatory parameter. Function: This command returns all configurations of an agent, including permissions, tools, model parameters, and prompts. It is used to debug abnormal agent behavior.
codearts debug paths	Definition: displays global paths. Function: This command returns core directories, including the data directory, configuration directory, cache directory, or status directory. It is used to troubleshoot issues such as missing files and permission issues.
codearts debug wait	Definition: waits indefinitely (for debugging). To exit, press the shortcut key (Windows: Ctrl+C; Mac: ⌘+C). Function: This command keeps a process running so that a debugger can be attached to the process for debugging.

Table 15-8 codearts debug rg subcommands

Command	Description
codearts debug rg tree	Definition: displays the file directory tree using ripgrep. With this command, you can: ● View the file structure using ripgrep and check the directories or files that are included or excluded. ● Check whether the .gitignore and .ignore rules take effect and why some files cannot be found. Typical use case: You can use this command when you suspect that project files were accidentally ignored and want to confirm the scanning scope of ripgrep.
codearts debug rg files	Definition: lists all files using ripgrep. With this command, you can: ● List all files within the scanning scope of ripgrep (affected by ignore rules). ● Quickly compare actual files with the files visible to ripgrep to locate the issue that files are not indexed. Typical use case: You can use this command when you cannot find a file in CodeArts Agent CLI and want to check whether the file has been ignored by ripgrep.
codearts debug rg search <pattern>	Definition: searches for file content using ripgrep. With this command, you can: ● Directly call ripgrep to search for file content and verify the underlying behavior of the search function in CodeArts Agent CLI. ● Compare the search results on CodeArts Agent CLI with those obtained using native ripgrep to determine whether an issue occurs in the frontend or backend. Typical use case: When search results are missing or matching rules are abnormal, you can run this command if you want to use native ripgrep for comparison and verification.

Table 15-9 codearts debug file subcommands

Command	Description
codearts debug file read <path>	Definition: reads and outputs the content of a specified file in JSON format. With this command, you can: • Directly view the content of files read by CodeArts Agent CLI, especially configuration files and status files. • Troubleshoot the following issues: A file exists but cannot be read; empty content is read; the format of the file read is incorrect. • Check whether the file encoding and JSON formats are correct.
codearts debug file status	Definition: displays the status of a file system. With this command, you can: • Check the statuses and permissions of files in the current directory or workspace (such as symbolic links and read-only permissions). • Troubleshoot issues such as insufficient file permissions and unavailable paths. • Check whether a file is locked or occupied by the system.
codearts debug file list <path>	Definition: lists all files and subdirectories in a specified directory of the current project. With this command, you can list files and subdirectories in a directory of a project.
codearts debug file search <query>	Definition: performs a fuzzy search for files in the current project by condition. With this command, you can quickly search for files that meet specified conditions.
codearts debug file tree [dir]	Definition: displays the tree structure of non-empty directories in a specified directory. This command displays the complete structure of a specified directory, allowing you to view levels of the directory.

Table 15-10 Parameters supported by codearts serve

Parameter	Description
--port	Server listening port. The default port is 4096. For example, you can run the following command to specify listening port 4096: `codearts serve --port 4096` After the service is started, you can open a new window and run the **codearts attach** command to connect to the server. For details, see **Table 15-1**.
--hostname	Server listening address. The default value is **127.0.0.1**. For example, you can run the following command to specify the server listening address 127.0.0.1: `codearts serve --hostname 127.0.0.1` After the service is started, you can open a new window and run the **codearts attach** command to connect to the server. For details, see **Table 15-1**.
--mdns	Enables discovery of the mDNS services with zero configuration on the LAN. Devices on the same LAN as CodeArts Agent CLI can be automatically scanned and can access CodeArts Agent CLI. The IP address of the local host does not need to be manually specified. For example, you can run the following command to start CodeArts Agent CLI on port 4096 of the local host and enable discovery of mDNS services on the LAN. Devices on the same LAN can automatically scan the current service and access the web management page and MCP interfaces using the .local domain name without the host IP address. `codearts serve --port 4096 --mdns`

--mdns-domain	This parameter must be used together with **--mdns**. It specifies the custom host name prefix of the mDNS LAN. The default value is **codearts.local**. If you do not specify **--mdns-domain**, the system uses the **codearts** prefix and combines it with the fixed mDNS suffix **.local** to generate the LAN domain name **codearts.local**.
	For example, the following command uses the default domain name **codearts.local**. The access address is http://codearts.local:4096.
	`codearts serve --port 4096 --mdns`
	The following command uses manually specified **--mdns-domain** to overwrite the default value. The domain name is dev-server.local, and the access address is http://dev-server.local:4096.
	`codearts serve --port 4096 --mdns --mdns-domain dev-server`

Parameter	Description

--cors <origin>	Addresses of external web pages that have permissions to access the service. If ***** is passed, all web pages can access the service. This is used for local development and debugging. If this parameter is not set, only pages of the service can call APIs. External web pages will be blocked by the browser.
	For example, you can run the following command to allow the local frontend page running on http://localhost:5173 to request APIs through port 4096:
	`codearts serve --port 4096 --cors http://localhost:5173`
	You can run the following command to allow access from multiple sources:
	`codearts serve --port 4096 \`
	`--cors http://localhost:5173 \`
	`--cors http://127.0.0.1:3000`
	You can run the following command to allow access from all sources. This command is used only for local debugging. **It may pose security risks and is not recommended for the production environment.**
	`codearts serve --port 4096 --cors *`

Table 15-11 Parameters supported by codearts attach

Parameter	Description
<url>	You can run this command to connect to a running server. You can perform the following steps to connect to a running server: 1. Configure temporary environment variables to specify the username and password for connecting to the server. • Run the following commands in Windows: `set CODEARTS_SERVER_USERNAME=Server username` `set CODEARTS_SERVER_PASSWORD=Server password` • Run the following commands in macOS or Linux: `export CODEARTS_SERVER_USERNAME="Server username"` `export CODEARTS_SERVER_PASSWORD="Server password"` 2. Run the following command to start the service on a terminal: `codearts serve` If the following information is displayed, the service is successfully started: `This service runs in a local development environment and is not a standard cloud service. It does not provide enterprise-grade security hardening or SLA guarantees. Do not expose the service address to the public internet or share it with others. This may lead to unauthorized access, code leaks, and other security risks.` `By continuing to use this service, you acknowledge and voluntarily assume these risks.` `It is advised to set up a reverse proxy to secure access. Configuration guide: https://support.huaweicloud.com/intl/en-us/codeartsagent_faq/codeartsagent_faq_0054.html` `codearts server listening on http://127.0.0.1:4096` 3. Run the following command on another terminal to connect to a specified server: `codearts attach ${CODEARTS_SERVER_URL} --password ${CODEARTS_SERVER_PASSWORD}` In the command, ${CODEARTS_SERVER_URL} indicates the URL of the server to be connected, and ${CODEARTS_SERVER_PASSWORD} indicates the password of the server. For example, you can run the following command to connect to the remote development server through port 4096 of the local host and access the development environment through password authentication: `codearts attach http://localhost:4096 -p yourpassword`
-p,--password	Server password
-c,--continue	Continues the last session.
-s,--session	Session ID
--fork	Fork session, which is used together with **--continue** or **--session**.

15.2 MCP Commands

codearts mcp commands are used to manage MCP servers, including adding, listing, authenticating, and debugging MCP servers.

codearts mcp add

This command adds an MCP server.

This example uses the project-level configuration in Windows as an example.

Step 1 Go to **D:/tmp/test_hc_cli**, right-click in the blank area, and select **Open Windows Terminal here**. In the displayed PowerShell window, run the following command and set parameters as prompted:

```
codearts mcp add
```

Step 2 Set **Location** parameters.

Figure 15-1 Setting Location parameters

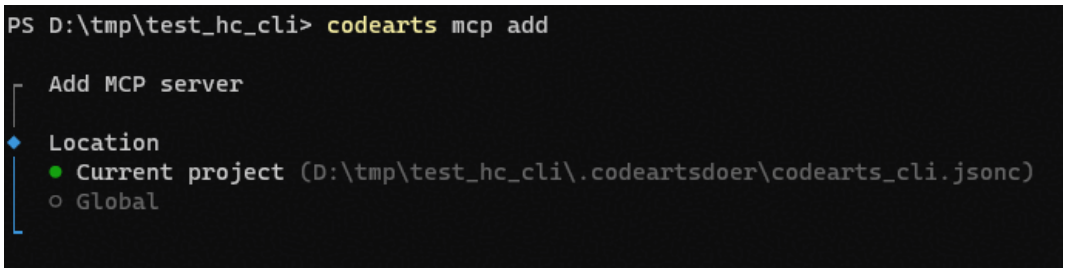


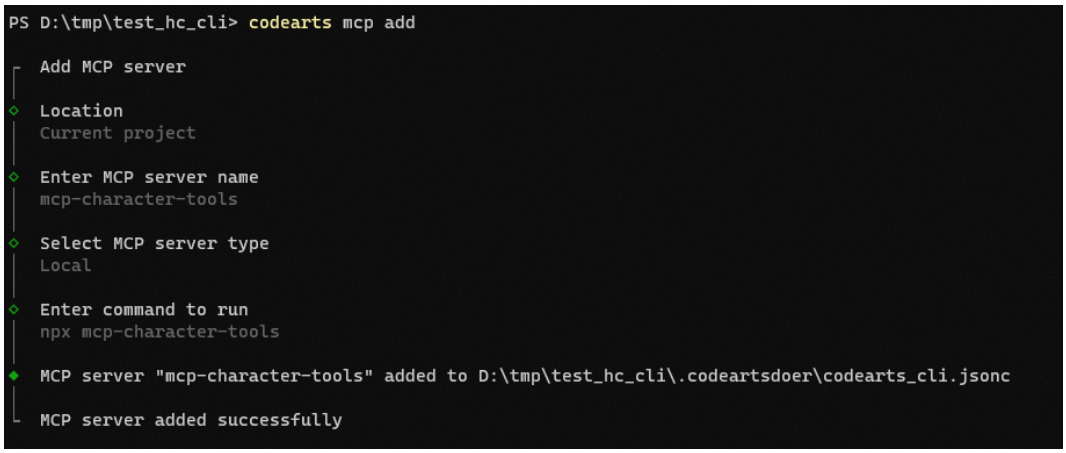
Table 15-12 lists the **Location** parameters.

Table 15-12 Location parameters

Parameter	Description
Current project	Project where the MCP server is to be added: *Project root directory*/**.codeartsdoer/codearts_cli.jsonc**
Global	Path where the MCP server is to be added: **~/.codeartsdoer/codearts_cli.jsonc**

Step 3 Set the **Enter MCP server name**, **Select MCP server type**, and **Enter command to run** parameters as prompted.

Figure 15-2 Configuration example



For details about the parameters, see **Table 15-13**.

Table 15-13 Other MCP server parameters

Parameter	Description
Enter MCP server name	Name of the MCP server
Select MCP server type	Running mode of the MCP server. The options are as follows: • **Local**: You can run a local command, such as **npx**, **uvx**, or **python** to start the server. The communication mode is stdio. • **Remote**: You need to enter the URL of the remote MCP server. CodeArts Agent CLI connects to the server over HTTP or SSE.
Enter command to run	Command for running the MCP server

Step 4 Run **codearts mcp list** to check whether the configured MCP server is available.

----End

codearts mcp list

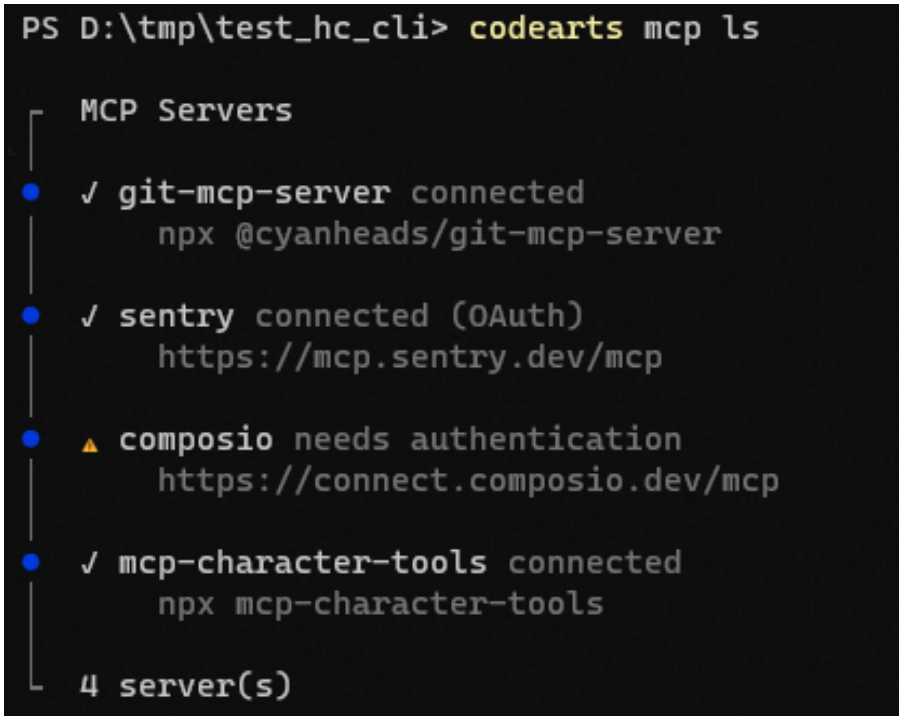
This command lists all configured MCP servers and their connection statuses.

For example, you can run the following command to view the MCP servers configured in the **D:/tmp/test_hc_cli** project directory and their connection statuses:

```
codearts mcp list
```

There are four MCP servers in the current directory. The **git-mcp-server**, **sentry**, and **mcp-character-tools** servers are connected, and the **composio** server is pending authentication.

Figure 15-3 Example



The following table describes the MCP server statuses.

Table 15-14 MCP server statuses

Icon	Status
✓	Connected
	Pending authentication
○	Uninitialized or disabled
✗	Failed

codearts mcp auth list

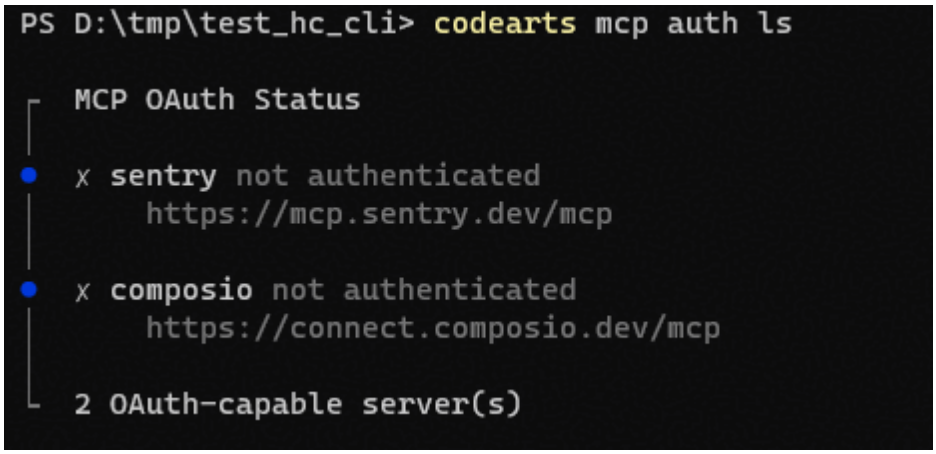
This command lists the MCP servers that support authentication and their authentication statuses.

For example, you can run the following command in the **D:/tmp/test_hc_cli** project directory to view the MCP servers that support authentication and their statuses.

```
codearts mcp auth list
```

As shown in the following figure, two MCP servers in the current project are pending authentication.

Figure 15-4 Servers pending authentication



codearts mcp auth [name]

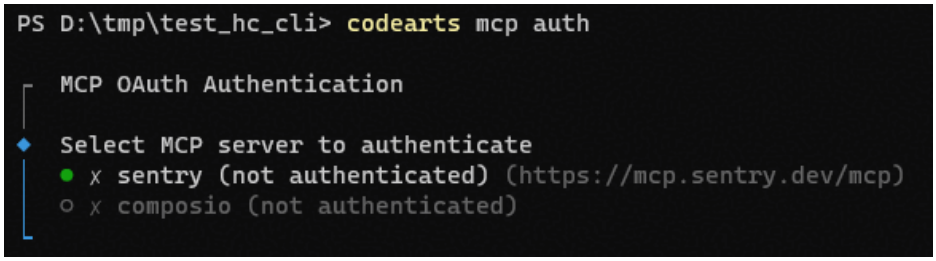
This command authenticates an MCP server.

Step 1 Run the following command to list the MCP servers that have not been authenticated. In the command output, select the MCP servers you want to authenticate and press **Enter** to start authentication.

```
codearts mcp auth
```

As shown in the following figure, there are two MCP servers pending authentication in the current project. Select the MCP server you want to authenticate and press **Enter** to start authentication.

Figure 15-5 MCP servers pending authentication



You can perform authentication by referring to the following example configuration:

```
"mcp": {
   "sentry": {
     "type": "remote",
     "url": "https://mcp.sentry.dev/mcp",
     "oauth": {}
   }
 }
```

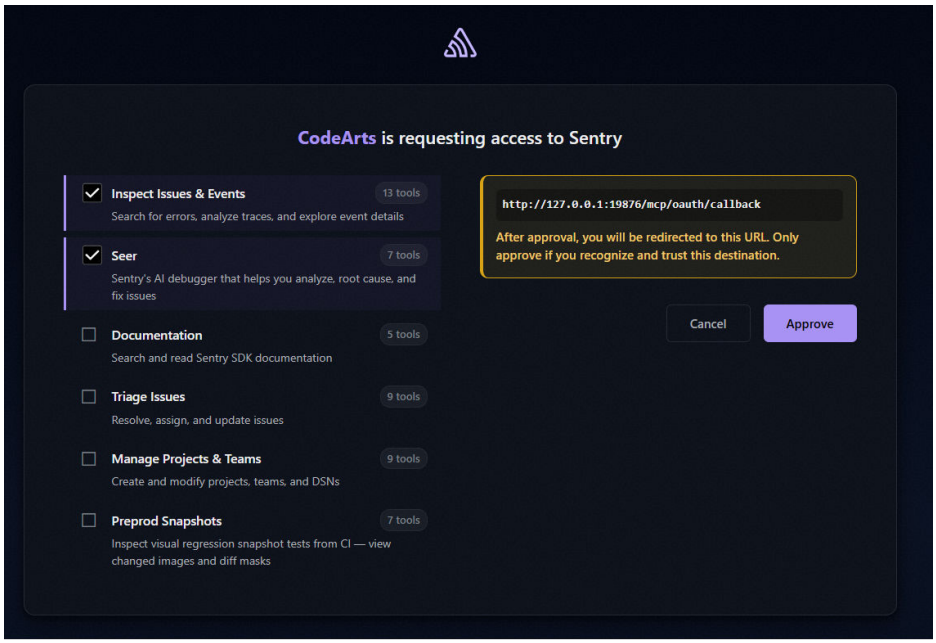
Step 2 After the configuration is complete, run the following command to open a browser window for authentication and connect CodeArts Agent CLI to your Sentry account. In the command, **sentry** is the MCP server name configured in the preceding step.

After the authentication is complete, the Sentry configuration is added to the **~/.codeartsdoer/cli-data/mcp-auth.json** file.


```
codearts mcp auth sentry
```

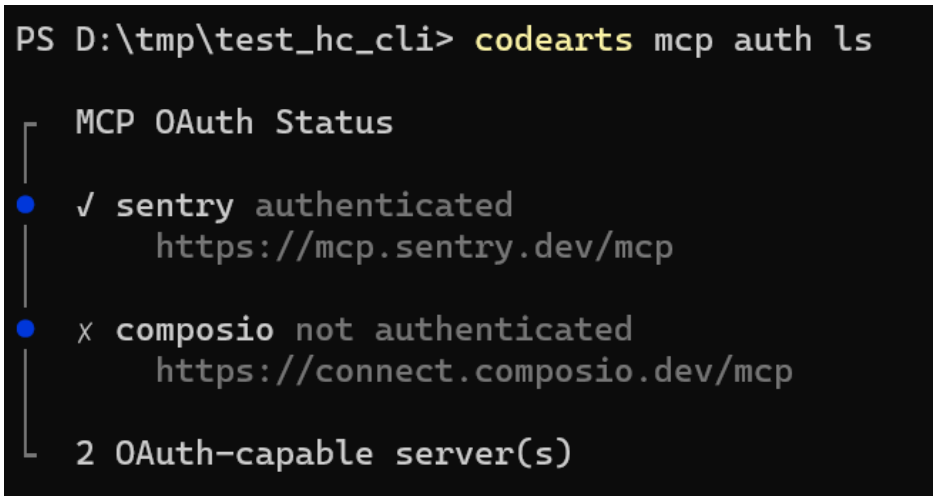
As shown in the following figure, the Sentry account authentication is complete.

Figure 15-6 Sentry account authentication



Step 3 Run the **codearts mcp auth ls** command again. The command output shows that the **sentry** server has been authenticated.

Figure 15-7 Successful authentication



----End

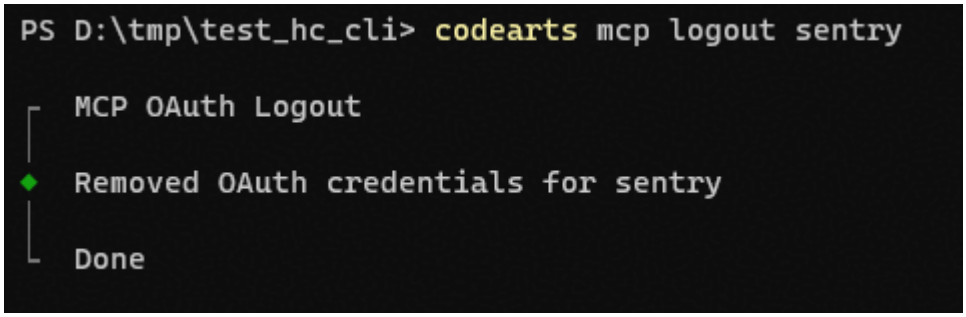
codearts mcp logout [name]

This command removes the authentication credentials of an MCP server.

For example, go to the **D:/tmp/test_hc_cli** project directory and run the following command to remove the authentication credentials of the project:

The MCP configuration in the **~/.codeartsdoer/cli-data/mcp-auth.json** file is deleted.

Figure 15-8 Removing authentication credentials



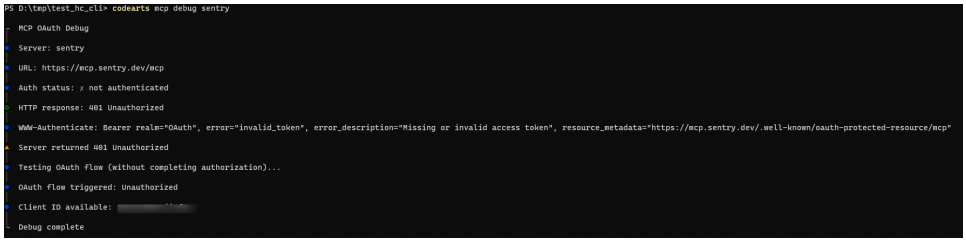
codearts mcp debug <name>

This command displays the authentication information and connection test result.

Example:

The following figure shows how to debug an MCP server for which OAuth authentication has not been performed.

Figure 15-9 Debugging



The following figure shows how to debug an MCP server for which OAuth authentication has been performed.

Figure 15-10 Debugging