

Claude Projects Enhanced CLI

User Guide & Instructions





Table of Contents

1. [Introduction](#)
 2. [Installation & Setup](#)
 3. [Getting Started](#)
 4. [Working with Artifacts](#)
 5. [Command Reference](#)
-

Introduction

Claude Projects Enhanced CLI is a powerful command-line tool that brings the Claude.ai website experience to your terminal. It provides project-based file management, comprehensive AI responses, and automatic code artifact generation.

Key Features

-  **Website-Parity Responses:** Get the same detailed, comprehensive responses as Claude.ai
-  **Automatic Artifacts:** Code improvements are automatically saved as separate files
-  **Project Isolation:** Each project maintains its own files and conversation history
-  **Persistent Storage:** All conversations and artifacts are saved for future reference

System Requirements

- Python 3.7 or higher
 - Internet connection for API calls
 - Anthropic API key
-

Installation & Setup

Step 1: Download the Script

Save `claude_projects_enhanced.py` to your desired directory.

Step 2: Install Dependencies

```
bash
```

```
pip install anthropic python-dotenv
```

Step 3: Configure API Key

Create a `.env` file in the same directory:

```
ANTHROPIC_API_KEY=your_api_key_here
```

Step 4: Run the CLI

```
bash
```

```
python claude_projects_enhanced.py
```

You should see:

```
🚀 Starting Enhanced Claude Projects CLI...  
This version aims for parity with Claude.ai website responses
```

```
┌ Claude Projects CLI - Enhanced Edition │  
│ Website-Parity Mode for Consistent Responses │  
└──────────────────────────────────────────┘
```

Getting Started

Creating Your First Project

1. Create a new project:

```
(claude) > create my_project  
✅ Created enhanced project: my_project
```

2. Add files to analyze:




```
(my_project) > add /path/to/your/code.py  
✅ Added: code.py
```

3. Chat with Claude about your files:

(my_project) > chat What does this code do and how can I improve it?


Understanding the Response


When you chat with Claude, you'll see:

-  Including 1 files from 'my_project' project
-  Using website-parity mode for comprehensive response...
-  Thinking deeply (website-style response)...

 Claude (opus) - Enhanced Response:

[Detailed analysis with code improvements...]

-  Created 5 code artifacts:
- ✓ ImprovedClass (python) → improvedclass_20250802_161611.python
 - ✓ OptimizedFunction (python) → optimizedfunction_20250802_161611.python
 - ...

 Enhanced conversation saved: conversation_20250802_161611.json

-  Response Metrics:
- Length: 15,234 characters
 - Words: 2,456 words
 - Code artifacts: 5

Working with Artifacts


What are Artifacts?

Artifacts are complete, production-ready code files that Claude automatically generates when providing improvements or solutions. They're extracted from the response and saved as separate files you can immediately use.

Viewing Artifacts

1. List all artifacts:

```
(my_project) > artifacts
```

 Artifacts in my_project:

Name	Size	Modified
------	------	----------

improvedclass_20250802_161611.python	2.5 KB	2025-08-02 16:16
optimizedfunction_20250802_161611.python	1.8 KB	2025-08-02 16:16

2. View a specific artifact:

```
(my_project) > view improvedclass_20250802_161611.python
```

3. View all files and artifacts together:

```
(my_project) > list
```

Exporting Artifacts


1. Export to a specific directory:

```
(my_project) > export artifacts ~/Desktop/my_improvements
```

 Exported 5 artifacts to: /Users/you/Desktop/my_improvements

2. Open project folder in file manager:

```
(my_project) > open_project_folder
```

 Opened project folder

Artifact File Structure

Your artifacts are stored in:

```
Claude_Projects/
├── my_project/
│   ├── files/      # Your original files
│   ├── artifacts/  # Generated improvements
│   │   ├── improvedclass_20250802_161611.python
│   │   ├── optimizedfunction_20250802_161611.python
│   │   └── ...
│   └── conversations/ # Chat history
```


Managing Artifacts


- **Clear all artifacts** (with confirmation):


(my_project) > clear_artifacts

- **Get a project summary:**

(my_project) > summary

 Project Summary: my_project

 Files: 3 (45.2 KB)

 Artifacts: 12 (156.8 KB)

 Conversations: 8

Command Reference

Project Management

Command	Description	Example
<code>create <name></code>	Create a new project	<code>create web_app</code>
<code>open <name></code>	Open existing project	<code>open web_app</code>
<code>projects</code>	List all projects	<code>projects</code>
<code>summary</code>	Show project statistics	<code>summary</code>

File Management

Command	Description	Example
<code>add <path></code>	Add file to project	<code>add ~/code/app.py</code>
<code>files</code>	List project files	<code>files</code>
<code>remove <name></code>	Remove file	<code>remove old_code.py</code>
<code>update</code>	Sync files from directory	<code>update</code>

Artifact Management

Command	Description	Example
<code>artifacts</code>	List all artifacts	<code>artifacts</code>
<code>view <name></code>	View file or artifact	<code>view improved_code.python</code>
<code>list</code>	Show files AND artifacts	<code>list</code>
<code>export artifacts <path></code>	Export artifacts	<code>export artifacts ~/Desktop</code>
<code>clear_artifacts</code>	Remove all artifacts	<code>clear_artifacts</code>

Conversation

Command	Description	Example
<code>chat <message></code>	Talk to Claude	<code>chat explain this algorithm</code>
<code>conversations</code>	List chat history	<code>conversations</code>
<code>model <name></code>	Switch model	<code>model opus</code>
<code>models</code>	List available models	<code>models</code>

Utilities

Command	Description
<code>open_project_folder</code>	Open in file manager
<code>help</code>	Show all commands
<code>help_artifacts</code>	Artifact help
<code>clear</code>	Clear screen
<code>exit</code> or <code>quit</code>	Exit CLI

Pro Tips

1. **Auto-completion:** Use TAB to complete commands
2. **Quick artifact access:** `view` works for both files and artifacts
3. **Batch operations:** Export all artifacts at once with `export artifacts`
4. **Model selection:** Use `opus` for best quality, `sonnet` for balance, `haiku` for speed

Common Workflows

Code Review Workflow:

```
bash
```

```
> create code_review
> add src/main.py src/utils.py src/models.py
> chat Review this code for potential issues and improvements
> export artifacts ~/Desktop/improvements
```

Learning Workflow:

```
bash

> create learning_python
> add confusing_code.py
> chat Explain this code in detail with examples
> view explanation_examples_*.python
```

Refactoring Workflow:

```
bash

> create refactor_project
> add legacy_code.py
> chat Refactor this code using modern best practices
> list # See all original files and new artifacts
> export artifacts ./refactored_code/
```

Troubleshooting

No API Key Error:

- Ensure `.env` file exists with `ANTHROPIC_API_KEY=your_key`

File Not Found:

- Use full paths or paths relative to current directory
- Check with `files` or `list` command

No Artifacts Created:

- Artifacts are only created for substantial code (100+ chars)
- Small snippets remain in the conversation

Can't Find Artifacts:

- Use `artifacts` to list all

- Use `open_project_folder` to browse manually
- Check `Claude_Projects/project_name/artifacts/`

For more information and updates, visit the project repository.