Using Stylus CLI

This guide will get you started using cargo stylus , a CLI toolkit to help developers manage, compile, deploy, and optimize their Stylus contracts efficiently.

This overview will help you discover and learn how to uses cargo stylus tools.

Installing cargo stylus

Cargo stylus is a plugin to the standard cargo tool for developing Rust programs.

Prerequisites

Rust toolchain Follow the instructions on Rust Lang's installation page to install a complete Rust toolchain (v1.81 or newer) on your system. After installation, ensure you can access the programs rustup ,rustc , and cargo from your preferred terminal application. Docker We will use the testnet, and some cargo stylus commands will require Docker to operate.

You can download Docker from Docker's website . Foundry's Cast Foundry's Cast is a command-line tool for interacting with your EVM contracts. Nitro devnode Stylus is available on Arbitrum Sepolia, but we'll use Nitro devnode, which has a pre-funded wallet, saving us the effort of wallet provisioning or running out of tokens to send transactions.

Install your devnode git clone https://github.com/OffchainLabs/nitro-devnode.git cd nitro-devnode Launch your devnode ./run-dev-node.sh

Installation

In your terminal, run:

cargo

install

--force cargo-stylus Add WASM (WebAssembly) as a build target for the specific Rust toolchain you are using. The below example sets your default Rust toolchain to 1.80 as well as adding the WASM build target:

rustup default 1.80 rustup target add wasm32-unknown-unknown --toolchain

1.80 You can verify the cargo stylus installation by running cargo stylus -V in your terminal, returning something like: stylus 0.5.6

Using cargo stylus

Cargo Stylus Commands Reference

Command Description Arguments Options Example Usage new Create a new Stylus project •name : Project name (required) •--minimal : Create a minimal contract cargo stylus new init Initialize a Stylus project in current directory

•--minimal : Create a minimal contract cargo stylus init --minimal export-abi Export a Solidity ABI

•--output : Output file (defaults to stdout) •--json : Write JSON ABI using solc cargo stylus export-abi --json activate Activate an already deployed contract •--address : Contract address to activate •--data-fee-bump-percent : Percent to bump estimated fee (default 20%) •--estimate-gas : Only estimate gas without sending transaction cargo stylus activate --address cache Cache contract using Stylus CacheManager •bid : Place bid on contract •status : Check contract status •suggest-bid : Get suggested minimum bid

cargo stylus cache bid --address check Check a contract

•--wasm-file : WASM file to check •--contract-address : Deployment address deploy Deploy a contract •--contract-address Where to deploy and activate the contract (defaults to a random address) •--estimate-gas : Only perform estimation •--no-verify : Skip reproducible container •--cargo-stylus-version : Version for Docker image •--source-files-for-project-hash : Path to source files to include in the project hash •--max-fee-per-gas-gwei : Optional max fee per gas in gwei units •--wasm-file : The WASM file to check (defaults to any found in the current directory) cargo stylus deploy --endpoint='http://localhost:8547' --private-key="" --estimate-gas verify Verify contract deployment •--deployment-tx : Hash of deployment transaction •--no-verify : Skip reproducible container •--cargo-stylus-version : Version for Docker image cgen Generate C code bindings •--input : Input file path •--out_dir : Output directory path replay Replay transaction in GDB •-t, --tx : Transaction to replay •-p, --project : Project path (default:.) •-u, --use-native-tracer : Use the native tracer instead of the JavaScript one (may not be available in the node) •-s, --stable-rust : Use stable Rust (note that nightly is needed to expand macros) cargo stylus replay --tx trace Trace a transaction •--tx : Transaction hash •--endpoint : RPC endpoint •--project : Project path •--use-native-tracer :

Use native tracer

Common options

These options are available across multiple commands:

Option Description --endpoint Arbitrum RPC endpoint (defaulthttp://localhost:8547) --verbose Print debug info --source-files-for-project-hash Paths to source files for project hash --max-fee-per-gas-gwei Optional max fee per gas in gwei

Authentication options

Available for commands involving transactions:

Option Description --private-key-path Path to file containing hex-encoded private key --private-key Private key as hex string (exposes to shell history) --keystore-path Path to Ethereum wallet keystore file --keystore-password-path Keystore password file path

How-tos

Topic Description Learn how to optimize WASM binaries Thecargo-stylus tool allows you to optimize WebAssembly (WASM) binaries, ensuring that your contracts are as efficient as possible. Debug Stylus transactions A guide to debugging transactions, helping you identify and fix issues. Gain insights into your Stylus contracts by debugging transactions. Verify contracts Ensure that your Stylus contracts are correctly verified. Step-by-step instructions on how to verify your contracts usingcargo-stylus . Run a Stylus dev node Learn how to run a local Arbitrum dev node to test your Stylus contracts.

Additional resources

Troubleshooting

: solve the most common issues.

cargo-stylus repository

: consult cargo stylus' source code. Edit this page Last updatedonJan 27, 2025 Previous Use Rust Crates Next Debug transactions