# Quickstart: Your First Contract

Write an ERC-20 token, test it, deploy it to the Neo Smart Chain testnet and verify it, using Foundry.

| VERSION | LAST UPDATED | STATUS |
| --- | --- | --- |
| 0.1 | November 17, 2026 | Draft |

## 1. What you will build

You will create a simple ERC-20 token with an owner who can mint, write tests for it, deploy it to the Neo Smart Chain testnet and publish its source code on the block explorer. The same steps apply to any other contract.

You need Git, a terminal and a code editor. This guide uses Foundry. If you prefer Hardhat, the deployment guide shows the equivalent configuration.

## 2. Step 1: Add the network to your wallet

Open your wallet, choose to add a custom network and enter the values from the network table in the developer overview: chain ID [chain ID], RPC URL [RPC URL], currency symbol [native currency symbol] and explorer [explorer URL].

Web applications can ask the wallet to add the network for the user with the standard wallet_addEthereumChain request:

```
await window.ethereum.request({
method: 'wallet_addEthereumChain',
params: [
{
chainId: '0x' + Number(process.env.NEXT_PUBLIC_NSC_CHAIN_ID).toString(16),
chainName: 'Neo Smart Chain',
nativeCurrency: {
name: process.env.NEXT_PUBLIC_NSC_CURRENCY_NAME,
symbol: process.env.NEXT_PUBLIC_NSC_CURRENCY_SYMBOL,
decimals: 18,
},
rpcUrls: [process.env.NEXT_PUBLIC_NSC_RPC_URL],
blockExplorerUrls: [process.env.NEXT_PUBLIC_NSC_EXPLORER_URL],
},
```

```
],
})
```

## 3. Step 2: Get test funds

Request testnet funds from the faucet at [testnet faucet URL]. Test funds have no value and exist only to pay for gas while you develop. Never send real funds to a testnet address.

## 4. Step 3: Set up the project

Install Foundry, create a project and add OpenZeppelin Contracts:

```
curl -L https://foundry.paradigm.xyz | bash
foundryup

forge init neo-quickstart
cd neo-quickstart
forge install OpenZeppelin/openzeppelin-contracts
```

Then configure the compiler in foundry.toml. Set evm_version to the highest version the network supports, as listed in the network table. The value below is a conservative default that avoids newer opcodes.

```
[profile.default]
src = "src"
out = "out"
libs = ["lib"]
solc_version = "0.8.28"
evm_version = "paris"
optimizer = true
optimizer_runs = 200
remappings = [
"@openzeppelin/contracts/=lib/openzeppelin-contracts/contracts/",
]
```

## 5. Step 4: Write the contract

Create src/NeoToken.sol. It extends audited OpenZeppelin building blocks instead of re-implementing the standard: ERC-20 for the token, ERC-2612 permit for gasless approvals and Ownable for the minting right.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
```

```
import {ERC20Permit} from "@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";
import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";

contract NeoToken is ERC20, ERC20Permit, Ownable {
constructor(address initialOwner, uint256 initialSupply)
ERC20("Neo Example", "NEX")
ERC20Permit("Neo Example")
Ownable(initialOwner)
{
_mint(initialOwner, initialSupply);
}

function mint(address to, uint256 amount) external onlyOwner {
_mint(to, amount);
}
}
```

## 6. Step 5: Test it

Create test/NeoToken.t.sol. The fuzz test runs with many random amounts, which catches edge cases that hand-written examples miss.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import {Test} from "forge-std/Test.sol";
import {NeoToken} from "../src/NeoToken.sol";

contract NeoTokenTest is Test {
NeoToken token;
address owner = makeAddr("owner");
address alice = makeAddr("alice");

function setUp() public {
token = new NeoToken(owner, 1_000_000 ether);
}

function test_InitialSupplyGoesToOwner() public view {
assertEq(token.balanceOf(owner), 1_000_000 ether);
}

function test_OnlyOwnerCanMint() public {
vm.prank(alice);
vm.expectRevert();
token.mint(alice, 1 ether);
}

function testFuzz_Transfer(uint256 amount) public {
amount = bound(amount, 0, 1_000_000 ether);
vm.prank(owner);
token.transfer(alice, amount);
assertEq(token.balanceOf(alice), amount);
```

```
}
}
```

```
forge test -vv
```

## 7. Step 6: Deploy to the testnet

Import your deployer key into an encrypted Foundry keystore so it never sits in plain text in your shell history or a .env file:

```
cast wallet import deployer --interactive
```

Create script/Deploy.s.sol:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import {Script} from "forge-std/Script.sol";
import {NeoToken} from "../src/NeoToken.sol";

contract Deploy is Script {
function run() external returns (NeoToken token) {
address owner = vm.envAddress("OWNER");
vm.startBroadcast();
token = new NeoToken(owner, 1_000_000 ether);
vm.stopBroadcast();
}
}
```

Set your environment values and broadcast the deployment:

```
export NSC_RPC_URL=<testnet RPC URL>
export NSC_CHAIN_ID=<testnet chain ID>
export OWNER=<your wallet address>

forge script script/Deploy.s.sol:Deploy \
--rpc-url $NSC_RPC_URL \
--account deployer \
--broadcast
```

The output lists the deployed contract address. Open it on the explorer at [explorer URL] to confirm the deployment.

## 8. Step 7: Verify the source code

Verification publishes your source code on the explorer so anyone can read it and confirm it matches the deployed bytecode. Set NSC_VERIFIER to the verifier type the explorer uses (etherscan or blockscout) and NSC_VERIFIER_URL to its API URL, both listed at [verifier details].

```
forge verify-contract <CONTRACT_ADDRESS> \
src/NeoToken.sol:NeoToken \
--chain-id $NSC_CHAIN_ID \
--verifier $NSC_VERIFIER \
--verifier-url $NSC_VERIFIER_URL \
--constructor-args $(cast abi-encode \
"constructor(address,uint256)" $OWNER $(cast to-wei 1000000))
```

## 9. Step 8: Interact with the contract

```
# Read a balance
cast call <CONTRACT_ADDRESS> "balanceOf(address)(uint256)" $OWNER \
--rpc-url $NSC_RPC_URL

# Send 10 tokens
cast send <CONTRACT_ADDRESS> "transfer(address,uint256)" <RECIPIENT> \
$(cast to-wei 10) --rpc-url $NSC_RPC_URL --account deployer
```

Next steps: read the security guide before deploying anything that will hold value, and the token listing guide when you are ready to publish documentation for your token.

This document is for information only and is not financial, legal or tax advice. Highlighted items in square brackets are still to be confirmed by the issuer. The latest version is always available at neosmartchain.com/developers/quickstart