

How to test and deploy smart contracts on Remix IDE?

1. Go to the Remix online IDE.

<https://remix.ethereum.org/>

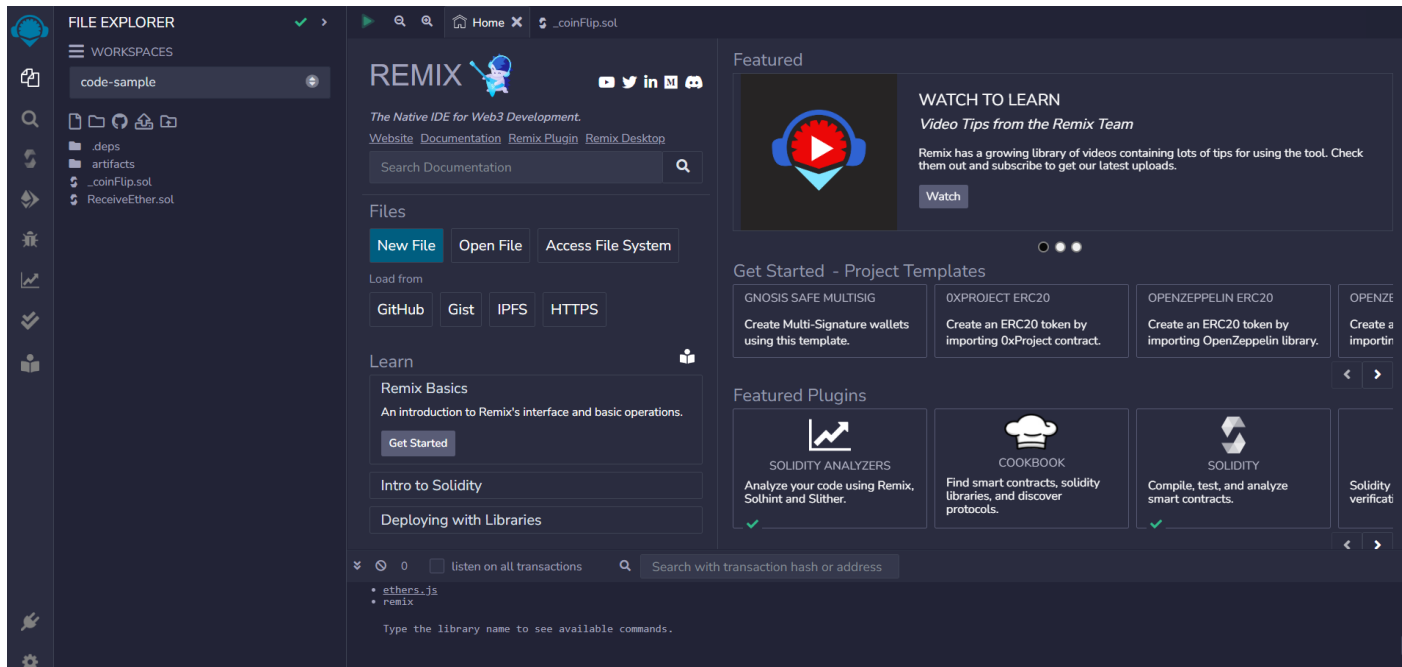


Fig. 1 Remix IDE

2. Write a Smart Contract

Create a Solidity file. The extension used for solidity language in which our smart contract is being written is '.sol'

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.7.0 <0.9.0;
```

```
contract CoinFlip {
    address public sender;
    address public player1;
    address public player2;
    uint256 public balance;

    constructor(address _player1, address _player2) {
        sender = msg.sender;
        player1 = _player1;
        player2 = _player2;
        balance = 0;
    }

    function addEther() external payable {
        require(msg.sender == sender, "Only the sender can add ether.");
        balance += msg.value;
    }

    function distribute() external {
        require(msg.sender == sender, "Only the sender can distribute.");

        uint256 lastDigit = block.timestamp % 10;

        if (lastDigit % 2 == 0) {
            payable(player1).transfer(balance);
        } else {
            payable(player2).transfer(balance);
        }
    }
}
```

```
}  
  
    balance = 100;  
}  
}
```

3. Compile the contract

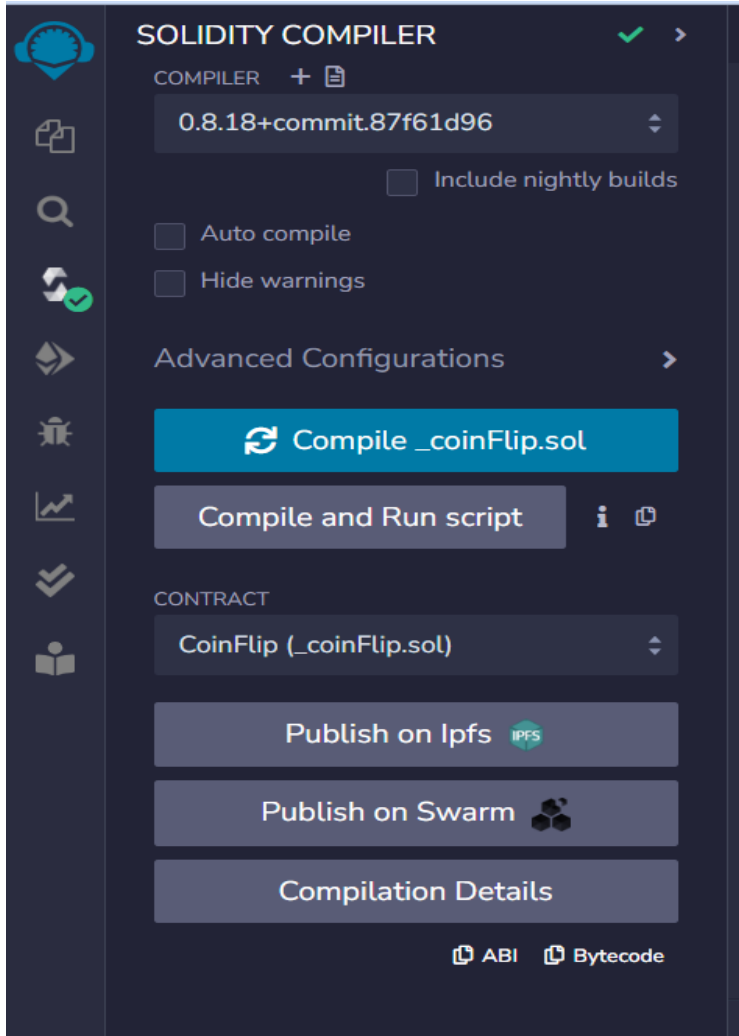


Fig. 2 Compiling the contract

4. Deploy the contract

DEPLOY & RUN TRANSACTIONS ✓ >

ENVIRONMENT

Remix VM (Shanghai)

VM

ACCOUNT

0x5B3...eddC4 (95.999999%)

GAS LIMIT

3000000

VALUE

0 Ether

CONTRACT

CoinFlip - _coinFlip.sol

evm version: paris

DEPLOY

_PLAYER1: 0x5ba565c557eaeed500514131

_PLAYER2: 0xc8ce5c867a4da3a540d6b2b5a

Calldata Parameters

☐ Publish to IPFS

At Address Load contract from Address

Fig. 3 Deploying the contract

Head to the Deploy and run transactions section.

For our coin flip contract, we have 2 players and a sender. When deploying the contract, we will need to select an address for the Sender, and the addresses for players 1 and 2. The provided accounts are just test ids and are only meant for testing purposes.

Click on transact to deploy the contract in the test environment!

5. Test the contract.

To test this contract we will need to send test Ether from the Sender's account to the balance of the smart contract. To do this, I have selected 4 Ethers and clicked 'addEther' function to add this balance (see fig 4).

When I click on the 'balance', I see 4 ETH as the current balance. Next, the 'distributeEther' function will randomly select either a 0 or a 1 and distribute the balance to either player 1 or 2, based on which play wins. The last figure shows how the sender's balance has been reduced and 4 ETH has been added to Player 1's account.

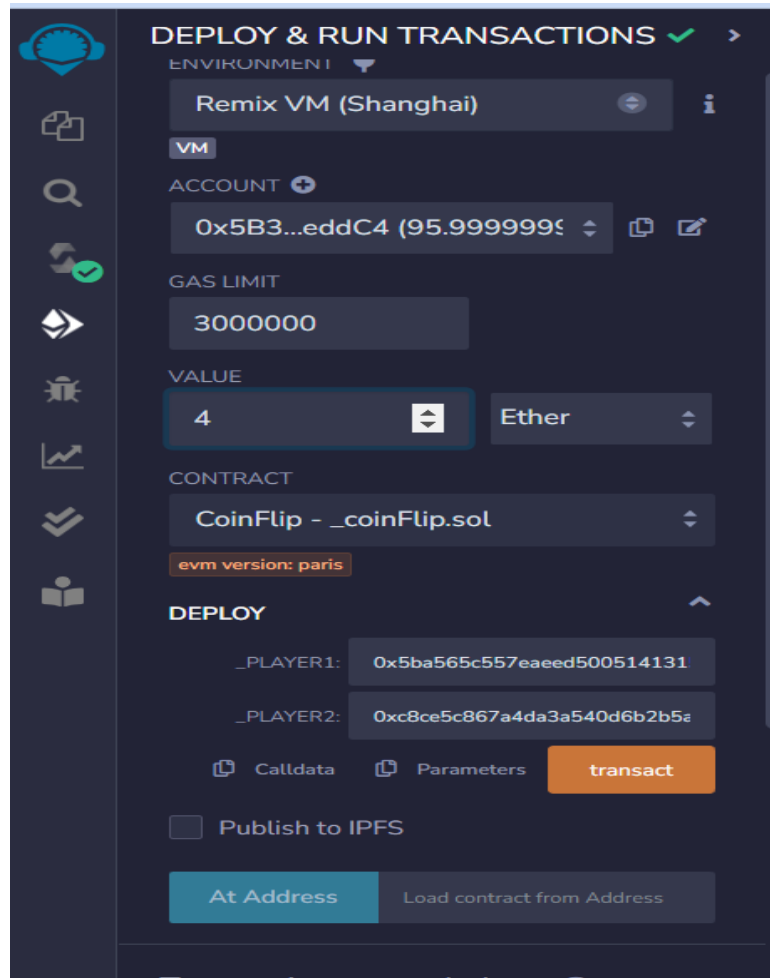


Fig. 4 Adding ETH to the contract

COINFLIP AT 0XD91...39138 (MEM)

Balance: 4 ETH

addEther

distributeEther

balance

0: uint256: 4000000000000000000

player1

player2

0x5B3...eddC4 (95.999999999999500449 ether)

0xAb8...35cb2 (100 ether)

0x4B2...C02db (100 ether)

0x787...cabaB (100 ether)

0x617...5E7f2 (100 ether)

0x17F...8c372 (100 ether)

0x5c6...21678 (100 ether)

0x03C...D1Ff7 (100 ether)

0x1aE...E454C (100 ether)

0x0A0...C70DC (100 ether)

0xCA3...a733c (100 ether)

0x147...C160C (100 ether)

0x4B0...4D2dB (100 ether)

0x583...40225 (100 ether)

0xD8...92148 (100 ether)

0x5Ba...8090E (104 ether)

0x548...D39d7 (100 ether)

0xC8c...78b5b (100 ether)

COINFLIP AT 0XD91...39138 (MEM)

Balance: 0 ETH

addEther

distributeEther

balance

0: uint256: 0

player1

player2

Fig. 7 Contract balance after distribution

By following these steps, we have successfully tested a smart contract.