Hardhat

Hardhat is used through a local installation in your project. This way your environment will be reproducible, and you will avoid future version conflicts.

To install it, you need to create an npm project by going to an empty folder, running npm init, and following its instructions. You can use another package manager, like yarn, but we recommend you use npm 7 or later, as it makes installing Hardhat plugins simpler.

Once your project is ready, you should run

npm 7+
npm 6
yarn

```
npm install --save-dev hardhat
```

To use your local installation of Hardhat, you need to use npx to run it (i.e. npx hardhat).

# Quick Start
TIP

If you are using Windows, we strongly recommend using WSL 2 to follow this guide.

We will explore the basics of creating a Hardhat project with a sample contract, tests of that contract, and a script to deploy it.

To create the sample project, run npx hardhat in your project folder:

```
$ npx hardhat
888    888                      888 888               888
888    888                      888 888               888
888    888                      888 888               888
8888888888  8888b.  888d888 .d88888 88888b.   8888b.  888888
888    888     "88b 888P"  d88" 888 888 "88b     "88b 888
888    888 .d888888 888    888  888 888  888 .d888888 888
888    888 888  888 888    Y88b 888 888  888 888  888 Y88b.
888    888 "Y888888 888     "Y88888 888  888 "Y888888  "Y888

👷 Welcome to Hardhat v2.9.9 👷

? What do you want to do? …
```

```
❯ Create a JavaScript project
  Create a TypeScript project
  Create an empty hardhat.config.js
  Quit
```
Let’s create the JavaScript or TypeScript project and go through these steps to compile, test and deploy the sample contract. We recommend using TypeScript, but if you are not familiar with it just pick JavaScript.

#Running tasks
To first get a quick sense of what's available and what's going on, run npx hardhat in your project folder:

```
$ npx hardhat
Hardhat version 2.9.9

Usage: hardhat [GLOBAL OPTIONS] <TASK> [TASK OPTIONS]

GLOBAL OPTIONS:

  --config              A Hardhat config file.
  --emoji               Use emoji in messages.
  --help                Shows this message, or a task's help if its name is provided
  --max-memory          The maximum amount of memory that Hardhat can use.
  --network             The network to connect to.
  --show-stack-traces   Show stack traces.
  --tsconfig            A TypeScript config file.
  --verbose             Enables Hardhat verbose logging
  --version             Shows hardhat's version.


AVAILABLE TASKS:

  check                 Check whatever you need
  clean                 Clears the cache and deletes all artifacts
  compile               Compiles the entire project, building all artifacts
  console               Opens a hardhat console
  coverage              Generates a code coverage report for tests
  flatten               Flattens and prints contracts and their dependencies
  help                  Prints this message
  node                  Starts a JSON-RPC server on top of Hardhat Network
  run                   Runs a user-defined script after compiling the project
  test                  Runs mocha tests
  typechain             Generate Typechain typings for compiled contracts
  verify                Verifies contract on Etherscan
```

To get help for a specific task run: npx hardhat help [task]
The list of available tasks includes the built-in ones and also those that came with any installed plugins. npx hardhat is your starting point to find out what tasks are available to run.

#Compiling your contracts
Next, if you take a look in the contracts/ folder, you'll see Lock.sol:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.9;

// Uncomment this line to use console.log
// import "hardhat/console.sol";

contract Lock {
   uint public unlockTime;
   address payable public owner;

   event Withdrawal(uint amount, uint when);

   constructor(uint _unlockTime) payable {
      require(
         block.timestamp < _unlockTime,
         "Unlock time should be in the future"
      );

      unlockTime = _unlockTime;
      owner = payable(msg.sender);
   }

   function withdraw() public {
      // Uncomment this line, and the import of "hardhat/console.sol", to print a log in your
terminal
      // console.log("Unlock time is %o and block timestamp is %o", unlockTime,
block.timestamp);

      require(block.timestamp >= unlockTime, "You can't withdraw yet");
      require(msg.sender == owner, "You aren't the owner");

      emit Withdrawal(address(this).balance, block.timestamp);

      owner.transfer(address(this).balance);
   }
}
```

To compile it, simply run:

```
npx hardhat compile
```

If you created a TypeScript project, this task will also generate TypeScript bindings using TypeChain.

#Testing your contracts
Your project comes with tests that use Mocha, Chai, and Ethers.js.

If you take a look in the test/ folder, you'll see a test file:

TypeScript
JavaScript

```
const {
  time,
  loadFixture,
} = require("@nomicfoundation/hardhat-network-helpers");
const { anyValue } = require("@nomicfoundation/hardhat-chai-matchers/withArgs");
const { expect } = require("chai");

describe("Lock", function () {
  // We define a fixture to reuse the same setup in every test.
  // We use loadFixture to run this setup once, snapshot that state,
  // and reset Hardhat Network to that snapshot in every test.
  async function deployOneYearLockFixture() {
    const ONE_YEAR_IN_SECS = 365 * 24 * 60 * 60;
    const ONE_GWEI = 1_000_000_000;

    const lockedAmount = ONE_GWEI;
    const unlockTime = (await time.latest()) + ONE_YEAR_IN_SECS;

    // Contracts are deployed using the first signer/account by default
    const [owner, otherAccount] = await ethers.getSigners();

    const Lock = await ethers.getContractFactory("Lock");
    const lock = await Lock.deploy(unlockTime, { value: lockedAmount });

    return { lock, unlockTime, lockedAmount, owner, otherAccount };
  }

  describe("Deployment", function () {
    it("Should set the right unlockTime", async function () {
      const { lock, unlockTime } = await loadFixture(deployOneYearLockFixture);
      expect(await lock.unlockTime()).to.equal(unlockTime);
    });

    it("Should set the right owner", async function () {
      const { lock, owner } = await loadFixture(deployOneYearLockFixture);

      expect(await lock.owner()).to.equal(owner.address);
    });

    it("Should receive and store the funds to lock", async function () {
      const { lock, lockedAmount } = await loadFixture(
        deployOneYearLockFixture
      );

      expect(await ethers.provider.getBalance(lock.address)).to.equal(
        lockedAmount
      );
    });

    it("Should fail if the unlockTime is not in the future", async function () {
      // We don't use the fixture here because we want a different deployment
      const latestTime = await time.latest();
      const Lock = await ethers.getContractFactory("Lock");
      await expect(Lock.deploy(latestTime, { value: 1 })).to.be.revertedWith(
        "Unlock time should be in the future"
      );
    });
  });

  describe("Withdrawals", function () {
    describe("Validations", function () {
      it("Should revert with the right error if called too soon", async function () {
        const { lock } = await loadFixture(deployOneYearLockFixture);

        await expect(lock.withdraw()).to.be.revertedWith(
          "You can't withdraw yet"
        );
      });

      it("Should revert with the right error if called from another account", async function () {
        const { lock, unlockTime, otherAccount } = await loadFixture(
          deployOneYearLockFixture
        );
        // We can increase the time in Hardhat Network
        await time.increaseTo(unlockTime);

        // We use lock.connect() to send a transaction from another account
        await expect(lock.connect(otherAccount).withdraw()).to.be.revertedWith(
          "You aren't the owner"
        );
      });

      it("Shouldn't fail if the unlockTime has arrived and the owner calls it", async function () {
        const { lock, unlockTime } = await loadFixture(
          deployOneYearLockFixture
        );

        // Transactions are sent using the first signer by default
        await time.increaseTo(unlockTime);

        await expect(lock.withdraw()).not.to.be.reverted;
      });
    });

    describe("Events", function () {
      it("Should emit an event on withdrawals", async function () {
        const { lock, unlockTime, lockedAmount } = await loadFixture(
          deployOneYearLockFixture
        );

        await time.increaseTo(unlockTime);

        await expect(lock.withdraw())
          .to.emit(lock, "Withdrawal")
          .withArgs(lockedAmount, anyValue); // We accept any value as `when` arg
      });
    });

    describe("Transfers", function () {
      it("Should transfer the funds to the owner", async function () {
        const { lock, unlockTime, lockedAmount, owner } = await loadFixture(
          deployOneYearLockFixture
        );

        await time.increaseTo(unlockTime);
```

```
        await expect(lock.withdraw()).to.changeEtherBalances(
          [owner, lock],
          [lockedAmount, -lockedAmount]
        );
      });
    });
  });
});
```

You can run your tests with npx hardhat test:

TypeScript
JavaScript

```
$ npx hardhat test
Compiled 2 Solidity files successfully


  Lock
    Deployment
      ✔ Should set the right unlockTime (610ms)
      ✔ Should set the right owner
      ✔ Should receive and store the funds to lock
      ✔ Should fail if the unlockTime is not in the future
    Withdrawals
      Validations
        ✔ Should revert with the right error if called too soon
        ✔ Should revert with the right error if called from another account
        ✔ Shouldn't fail if the unlockTime has arrived and the owner calls it
      Events
        ✔ Should emit an event on withdrawals
      Transfers
        ✔ Should transfer the funds to the owner


  9 passing (790ms)
```

#Deploying your contracts
Next, to deploy the contract we will use a Hardhat script.

Inside the scripts/ folder you will find a file with the following code:

TypeScript
JavaScript

```
// We require the Hardhat Runtime Environment explicitly here. This is optional
// but useful for running the script in a standalone fashion through `node <script>`.
```

```
//
// You can also run a script with `npx hardhat run <script>`. If you do that, Hardhat
// will compile your contracts, add the Hardhat Runtime Environment's members to the
// global scope, and execute the script.
const hre = require("hardhat");

async function main() {
  const currentTimestampInSeconds = Math.round(Date.now() / 1000);
  const unlockTime = currentTimestampInSeconds + 60;

  const lockedAmount = hre.ethers.utils.parseEther("0.001");

  const Lock = await hre.ethers.getContractFactory("Lock");
  const lock = await Lock.deploy(unlockTime, { value: lockedAmount });

  await lock.deployed();

  console.log(
    `Lock with ${ethers.utils.formatEther(
      lockedAmount
    )}ETH and unlock timestamp ${unlockTime} deployed to ${lock.address}`
  );
}

// We recommend this pattern to be able to use async/await everywhere
// and properly handle errors.
main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

You can run it using npx hardhat run:

TypeScript
JavaScript

```
$ npx hardhat run scripts/deploy.js
Lock with 1 ETH deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

#Connecting a wallet or Dapp to Hardhat Network
By default, Hardhat will spin up a new in-memory instance of Hardhat Network on startup. It's also possible to run Hardhat Network in a standalone fashion so that external clients can connect to it. This could be a wallet, your Dapp front-end, or a script.

To run Hardhat Network in this way, run npx hardhat node:

```
$ npx hardhat node
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/
```

This will expose a JSON-RPC interface to Hardhat Network. To use it connect your wallet or application to http://127.0.0.1:8545.

If you want to connect Hardhat to this node, for example to run a deployment script against it, you simply need to run it using --network localhost.

To try this, start a node with npx hardhat node and re-run the deployment script using the network option:

TypeScript
JavaScript

```
npx hardhat run scripts/deploy.js --network localhost
```

Congrats! You have created a project and compiled, tested and deployed a smart contract.