

Getting Started as a Web3 Developer

This guide is meant to help you get started as a web3 developer using **JavaScript** aka JS as the web2 backend as the bridge between smart contracts and the frontend. JS is currently the most widely used language for smart contract development.

Contents

Software Prerequisites	2
Node.js	2
Visual Studio Code	2
Connecting VS Code to GitHub	3
Node Packages	4
Solc	4
Smart Contract Development Framework	4
Smart Contract Interaction	5
(Unit) Testing	5
Basic Terminal Commands	6

Software Prerequisites

Node.js

Download link: <https://nodejs.org/en/download/>

Choose the LTS (long term support) version. Installing Node.js includes **npm** and **npx** which is a CLI (command-line interface) tool that will be frequently used. Type “npm” in your local terminal to check if it’s installed.

Downloads


Latest LTS Version: 16.13.0 (includes npm 8.1.0)


Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS
Recommended For Most Users

Current
Latest Features


Windows Installer
node-v16.13.0-x86.msi


macOS Installer
node-v16.13.0.pkg


Source Code
node-v16.13.0.tar.gz

Windows Installer (.msi)
Windows Binary (.zip)
macOS Installer (.pkg)
macOS Binary (.tar.gz)
Linux Binaries (x64)
Linux Binaries (ARM)
Source Code

32-bit	64-bit
32-bit	64-bit
64-bit / ARM64	
64-bit	ARM64
64-bit	
ARMv7	ARMv8
node-v16.13.0.tar.gz	

npm: node package manager, used for installing package using **npm install [package-name]**, you can also **npm uninstall**

npx: node package execute, used for executing node packages using **npx [package-name] [action]**


Visual Studio Code


This will be your local code editor for smart contract development. You can choose any code editor, but I recommend using VS Code.


Download link: <https://code.visualstudio.com/download>

Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.


Windows
Windows 10, 11


.deb
Debian, Ubuntu
.rpm
Red Hat, Fedora, SUSE


Mac
macOS 10.11+

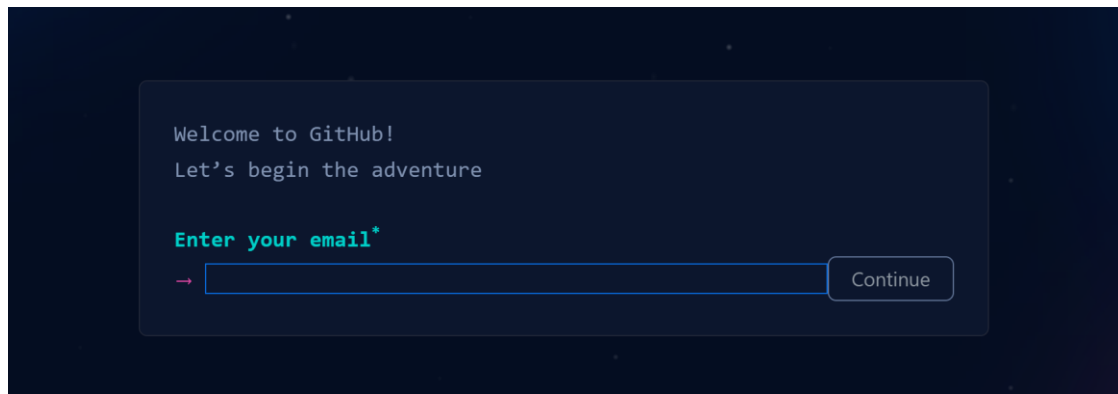
User Installer x64 x86 Arm64
System Installer x64 x86 Arm64
.zip x64 x86 Arm64
CLI x64 x86 Arm64

.deb x64 Arm32 Arm64
.rpm x64 Arm32 Arm64
.tar.gz x64 Arm32 Arm64
Snap Snap Store
CLI x64 Arm32 Arm64

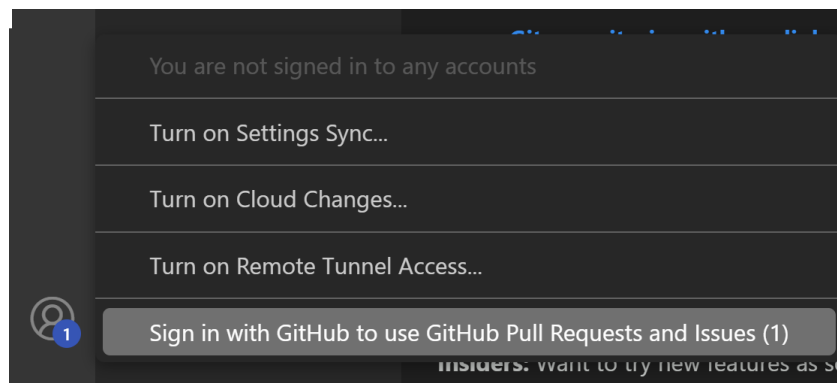
.zip Intel chip Apple silicon Universal
CLI Intel chip Apple silicon

Connecting VS Code to GitHub

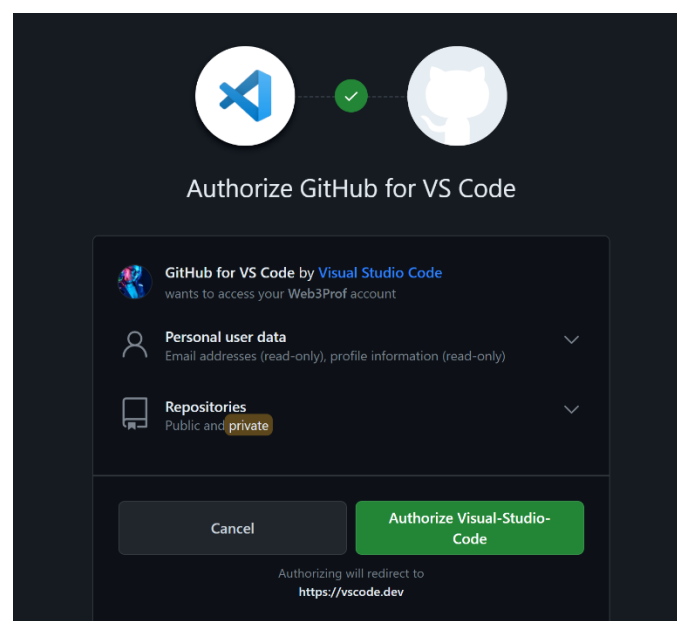
This is **optional**, but normally, developers own a GitHub which helps showcase their portfolio. If you haven't created a GitHub, sign up [here](#).



Click on the profile icon on the bottom left corner and select "Sign in with GitHub to use GitHub Pull Requests and Issues".



It will then trigger a new tab in your default browser to ask for authorization, choose "Authorize Visual-Studio-Code" and enter your password. Now, your GitHub should be connected to VS Code.



Node Packages

Node packages can be installed globally which will be applied to any project directory or specific to a project. Use **-S** or **--save** (for npm version before v5.0.0) for packages that are needed for the app to work properly. Use **-D** or **--save-dev** for packages that are only used for development phase and not required for deployment. All packages are added to **package.json** file. These flags will separate packages to **dependencies** and **devDependencies** section.

Solc

npm install -S solc@[version]

Solc is a Solidity compiler. This is required for compiling Solidity smart contracts. Mind the version. If using **Hardhat**, solc will be automatically downloaded. Set the version from **hardhat.config.js**.

```
module.exports = {
  solidity: "0.8.19",
};
```

Smart Contract Development Framework

These are the tools that'll help smart contract development. Choose either Truffle or Hardhat, but I prefer Hardhat.

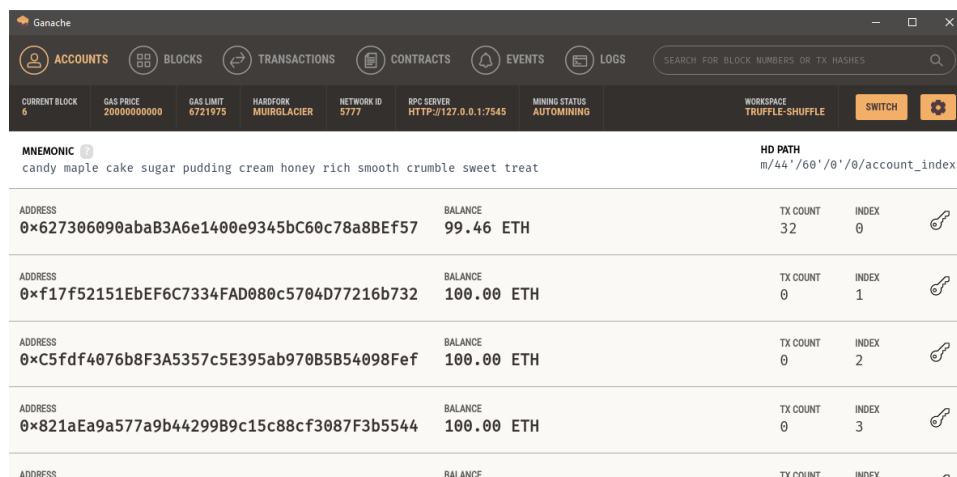
Truffle

npm install -D truffle

Developed by Consensys and usually pairs with **Ganache** for local blockchain. Check out [Truffle Suite](#) to setup the ultimate Truffle framework.

Ganache has [Ganache UI](#), the desktop application that shows graphical information about the local blockchain and might be preferred by new developers. There's also a command line version, Ganache CLI.

npm install -D ganache



The screenshot shows the Ganache UI interface. At the top, there's a navigation bar with tabs for ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below this is a status bar showing various network metrics like current block, gas price, gas limit, network ID, RPC server, and mining status. The main area displays a list of accounts with their addresses, balances, transaction counts, and indices. The first account has a balance of 99.46 ETH and 32 transactions. The other three accounts have a balance of 100.00 ETH and 0 transactions.

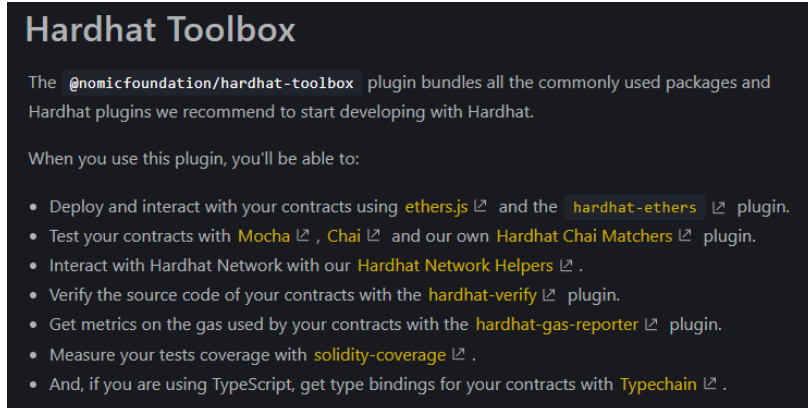
ADDRESS	BALANCE	TX COUNT	INDEX
0x627306090abaB3A6e1400e9345bC60c78a8BEf57	99.46 ETH	32	0
0xf17f52151EbEF6C7334FAD080c5704D77216b732	100.00 ETH	0	1
0xC5fdf4076b8F3A5357c5E395ab970B5B54098FeF	100.00 ETH	0	2
0x821aEa9a577a9b44299B9c15c88cf3087F3b5544	100.00 ETH	0	3

Hardhat

npm install -D hardhat

Developed by Nomic Labs and has a built-in Hardhat Network for local blockchain. Personally prefer Hardhat because it has the **Hardhat Toolbox**, a plugin that bundles all the commonly used packages for smart contract development.

`npm install -D @nomicfoundation/hardhat-toolbox`



Hardhat Toolbox

The `@nomicfoundation/hardhat-toolbox` plugin bundles all the commonly used packages and Hardhat plugins we recommend to start developing with Hardhat.

When you use this plugin, you'll be able to:

- Deploy and interact with your contracts using `ethers.js` and the `hardhat-ethers` plugin.
- Test your contracts with `Mocha`, `Chai` and our own `Hardhat Chai Matchers` plugin.
- Interact with Hardhat Network with our `Hardhat Network Helpers`.
- Verify the source code of your contracts with the `hardhat-verify` plugin.
- Get metrics on the gas used by your contracts with the `hardhat-gas-reporter` plugin.
- Measure your tests coverage with `solidity-coverage`.
- And, if you are using TypeScript, get type bindings for your contracts with `Typechain`.

Smart Contract Interaction

These are libraries that'll help interact to deployed smart contracts and blockchain via a local node or remote node. Choose either Web3 or Ethers, both are widely used. [Read this blog by OperReplay](#) to understand the difference between the two.

[Web3.js](#)

Web3.js is created by the Ethereum Foundation in 2015.

[Ethers.js](#)

Ethers.js is created by Richard Moore in 2016 as Web3.js's alternative.

(Unit) Testing

Unit testing means testing the smallest testable part of the code, usually useful when the code gets more complex. Assertion basically means confirming that the code works as expected. No package installation required for unit testing if using Hardhat.

[Waffle](#)

Waffle is a testing framework specifically designed for smart contracts based on Ethers.js and Mocha. It wraps Mocha and Chai. *I know, it's confusing, you don't really need to understand how these packages are related to each other as long as you can implement it.* [Waffle extends Chai's matchers](#) for smart contract testing such as BigNumber, address balance, and so on.

[Mocha](#)

Mocha is a testing framework. Mocha uses Node.js' built-in assert module.

[Chai](#)

Chai is an **assertion library** which means it provides tools called **matchers** for asserting values. It can work together with Mocha. Chai offers syntax options that can provide more readability.



Should	Expect	Assert
<pre>chai.should(); foo.should.be.a('string'); foo.should.equal('bar'); foo.should.have.lengthOf(3); tea.should.have.property('flavors') .with.lengthOf(3);</pre>	<pre>var expect = chai.expect; expect(foo).to.be.a('string'); expect(foo).to.equal('bar'); expect(foo).to.have.lengthOf(3); expect(tea).to.have.property('flavors') .with.lengthOf(3);</pre>	<pre>var assert = chai.assert; assert.typeOf(foo, 'string'); assert.equal(foo, 'bar'); assert.lengthOf(foo, 3); assert.property(tea, 'flavors'); assert.lengthOf(tea.flavors, 3);</pre>
Visit Should Guide	Visit Expect Guide	Visit Assert Guide

Basic Terminal Commands

The top 3 commands are usually enough. Tips: Press **Tab** on keyboard for filename autocomplete.

Command	Function	Syntax
<i>cd</i>	Change directory	cd [path]
<i>ls</i>	View directory (files & folders)	ls [path]
<i>mkdir</i>	Make directory (create folder)	mkdir [path-name]
<i>mv</i>	Move file	mv [file-name] [path-name]
<i>cp</i>	Copy file	cp [file-name] [path-name]
<i>rm</i>	Remove objects (files, folders)	rm [filename] rm -d [path]
<i>whatis</i>	Command description	whatis [command]
<i>man</i>	Command manual	man [command]
<i>clear</i>	Clear terminal	clear
<i>exit</i>	Exit terminal	exit

Here are some frequently used **npm** commands. **Opt** stands for optional.

Command	Function	Syntax
<i>init</i>	Initialize Node.js project Use -y to skip questions	npm init -y
<i>install</i>	Install Node package Use @[version] for specific version, opt	npm install [package]@[version]
<i>view</i>	View package detail Use versions to view all package versions, opt	npm view [package] versions
<i>list</i>	List all installed packages	npm list