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.

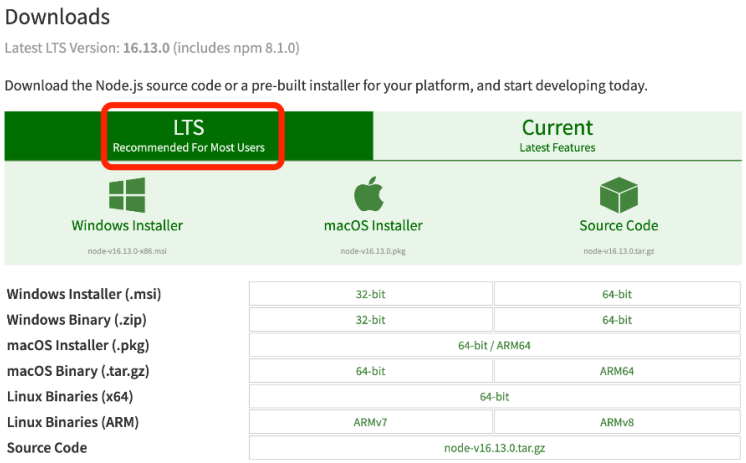
Contents

Software Prerequisites

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.



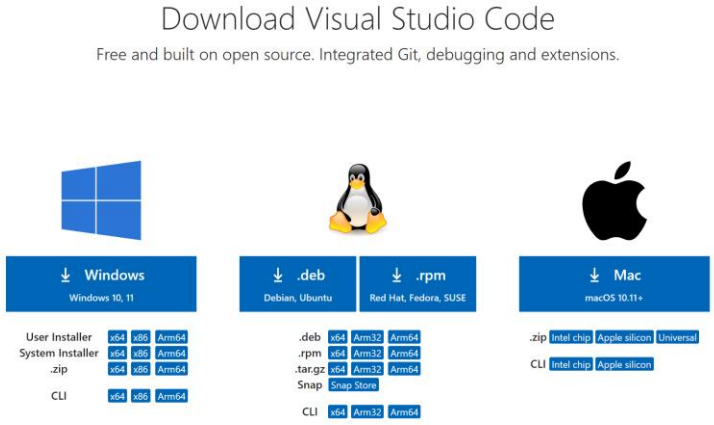
npm: node package manager, used for installing package using **npm install [package-name]**, you can also **npm uninstall**

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

Visual Studio Code

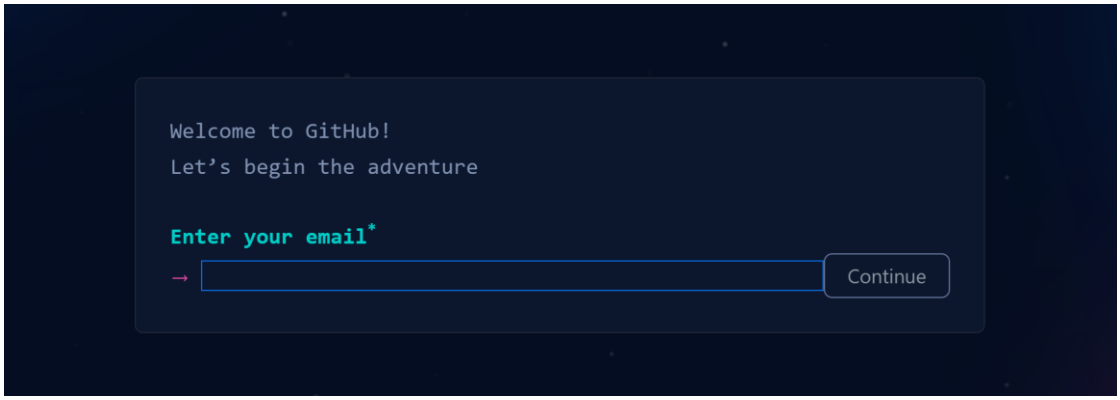
This will be your local code editor for smart contract development. You can choose any code editor, but I recommend using VS Code.

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

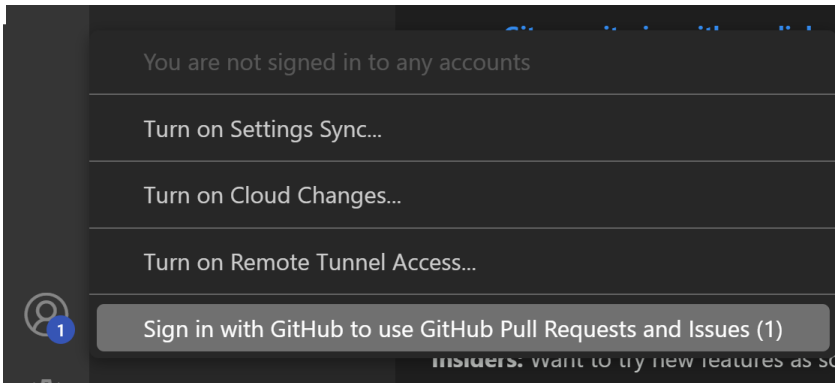


Connecting VS Code to GitHub

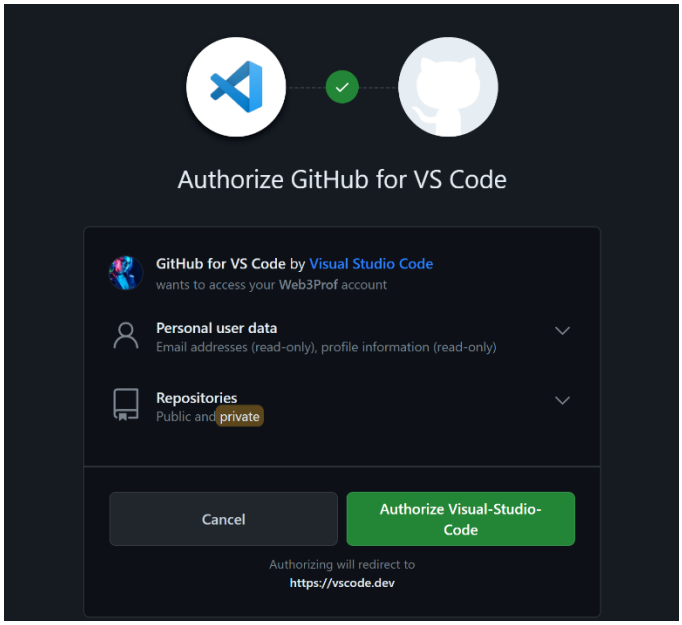
This is **optional**, but normally, developers own a GitHub which helps showcase their portfolio. If you haven't created a GitHub, sign up here.



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.

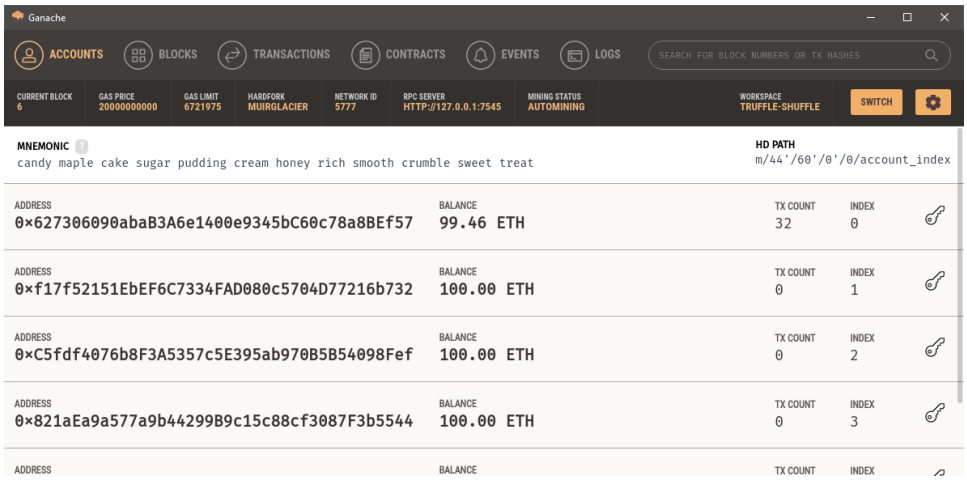
Truffle

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



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

```
chai.should();

foo.should.be.a('string');
foo.should.equal('bar');
foo.should.have.lengthOf(3);
tea.should.have.property('flavors')
  .with.lengthOf(3);
```

Visit Should Guide

Expect

```
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);
```

Visit Expect Guide

Assert

```
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);
```

Visit Assert Guide

Basic Terminal Commands

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

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

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

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