# README.md:

<p><img src="./BaseWordmarkBlue.svg"/></p>

Build On Base

Guides

These are a set of guides to help build on Base. Looking forward to seeing you onchain.

Builder Guides

Title	Description
Based	Learn how Base is built through technical documentation

ERC Guides

Title	Description
6551	How to deploy a token bound account

Bridge Guides

Title	Description
Native	How to bridge tokens to base

Troubleshooting

If you run into any troubles, please reach out for help on the Base Discord

License

All files within this repository are licensed under the Apache 2.0 License unless stated otherwise.

# README.md:

all your base are belong to you.

<p><img src="./base.jpeg"/></p>

This guide is intended to act as a tour of the Base ecosystem in order to help
with technical ramping to the protocol. It is intended to be a crash course moving

through major topics, providing links to articles and tutorials to help filter out the significant amount of noise one encounters when trying to learn how blockchain works for the first time. This is not exhaustive, but many of the resources referenced within the guide have additional content in their portals to go deeper once you have a functional understanding. All resources are free to access, and all time estimates are ScientificWildAssGuesses (SWAG).

The guide is intended to be read in the following order (but feel free to jump around as you see fit):

Article	Description
Blockchain Basics	Background material that will help with understanding how blockchains work
Ethereum	Ethereum is the Layer 1 that Base is built on, and understanding how it works is critical
Smart Contracts and Solidity	Smart contracts let us codify onchain contracts and transactions, with Solidity being the most popular language they're written in
Base	Learn how Base was built on top of the Bedrock OP Stack

# index.md:

Home | Previous

Base

<p><img src="./BaseWordmarkBlue.svg"/></p>

Now that we’ve covered core features of Ethereum as well as how Solidity and Smart Contract development works, we can dive into learning about Base. To get started here, read the Guide to Base which provides a well-rounded overview of the entire protocol. We’ll be breaking down the pieces of this guide so that you can fully understand how the protocol works.

Layer 2 Networks

Base is built as a Layer 2 network on top of Ethereum. With the popularity of Ethereum so high, demand has grown to where transaction speed has gone down and gas costs have shot up significantly. The solution to scaling Ethereum is using a Layer 2 blockchain, which is a network that fully functions on its own, but settles all transactions ultimately to the Ethereum network. Base is a specific implementation of L2 called an optimistic rollup. To better understand scaling and optimistic rollups, read the following three articles by Ethereum Foundation:

Title	Description
Scaling	How Ethereum has scaling issues and the mechanisms to fix them onchain and offchain (Layer 2 Networks)
Optimistic Rollups	How optimistic rollup networks work
Zero Knowledge Rollups	How ZK Proof rollup networks work

OP Bedrock

Base is built on top of the Bedrock upgrade to the Optimism (OP) Stack. To better understand Optimism Bedrock, we'll need to dive into Optimism's documentation.

Title	Description
Design Philosophy	Overall design philosophy on how Optimism was built with its core tenets
Rollup Protocol	The design of the OP Stack's rollup protocol
Bedrock Explainer	How OP Bedrock was designed, covering key information about the Sequencer

The Optimism repo contains the full Bedrock spec.

The following articles cover the Bedrock Data Flows:

Title	Description

Transaction Flow	How transactions flow through the OP Stack
Deposit Flow	How deposits work in OP Stack
Withdrawal Flow	How withdrawals work in OP Stack

Base being built on Bedrock makes it a part of the Superchain, a decentralized network of chains that share bridging, decentralized governance, upgrades, a communication layer and more. Optimism provides this explainer for their vision of the Superchain.

Base Specific Documentation

Base has its own doc site which helps to explain the specifics of what you need to know to build on the chain. Some highlights include:

Title	Description
Network Information	Network information for configuring developer environment endpoints
Key Contract Addresses	Addresses of key Smart Contracts deployed from OP Stack code
How Fees Work	How fees work on the Base network

Coinbase published a Guide to Base which highlights the core features and benefits of the network.

All code that's been open sourced related to Base can be found in this repo

# index.md:

Home | Next

Blockchain Basics

<p><img src="./whatisablockchain.png"/></p>

Blockchains are complex. The technology that makes them work is a result of decades of innovation across several disciplines. To understand how we got to where we are today, there needs to be some basic understanding of the underlying concepts. These articles are meant to be a relatively high level overview of the core concepts, and most are part of larger hubs of articles in the event you need to dive deeper into a specific topic.

Geeks for Geeks has a few sets of articles that are good starting points. This will give a background in how to understand the way blockchain tech works. Learn a basic understanding of computer networking (30 min)

This full hub of articles is worth going through at some point. The goal for now is to just learn some of the common terminology and reading the main intro article is sufficient. Everything on the blockchain is handled through networking distributed nodes, and having some understanding of computer networking is essential.

What is Peer to Peer Process (20 min)

The specific type of network that a blockchain runs on is Peer to Peer. This is from the blockchain knowledge series on Geeks for Geeks, which again is worth diving into fully at some point.

Blockchain Cryptography (20 min)

Blockchain networks only function through effective cryptography. The use of asymmetric keys and hashing is a core concept and referred to frequently throughout the Ethereum and Base documentation.

(Optional) Tree Data structure
(30 min+)

The core data structures used to store the blockchain data is built on types of tree data structures called tries. This is helpful from a deeper Computer Science background, but is not necessary to go deeper.

What is Blockchain Technology? (5 min)

This is a very high-level overview of blockchain as a concept. It helps pull together some of the things mentioned in the above fundamentals.

Finally a worthwhile read is the Bitcoin whitepaper (30 min). Bitcoin was the first cryptocurrency to launch as a functioning, decentralized blockchain. This article is significant as nothing that we are working towards now with Base would exist without Bitcoin.

# index.md:

Home | Previous | Next

Ethereum

<p><img width="61.4" height="100" src="./eth-diamond-purple.png"/></p>

Base is a layer 2 (L2) blockchain network built on top of Ethereum. In order to understand how the L2 rollup works, you need to have a good understanding of how Ethereum works.

To start you should read the original Ethereum whitepaper. A lot of the technology behind Ethereum has changed since then (most significantly the move to full Proof of Stake after the merge), but many of the core pieces of Ethereum as a computational platform are explained here. (1 hour)

Base Camp (the Base developer guide) has a good Intro video on Ethereum (8 min)

The Ethereum Foundation maintains a set of articles around the core functionality of Ethereum. These are all core readings to understand the protocol. Most of these articles are short and should take roughly 5 - 10 min each.

Title	Description
Intro to Ethereum	Introduction to the Ethereum Blockchain
Intro to Ether	Introduction to Ether Cryptocurrency
Introduction to Dapps	Introduction to Decentralized Applications
Web2 vs Web3	The differences between Web2 and Web3
Ethereum Accounts	How Accounts work on Ethereum
Transactions	What makes an Ethereum Transaction and how they work
Blocks	How Ethereum state organizes into Blocks
EVM (skip Opcodes)	How the Ethereum Virtual Machine powers computation in the protocol

Gas	How gas is used as fees in the network
Nodes and Clients	Overview of Ethereum Nodes and the clients used to run them
Client Diversity	Why there are multiple clients for running Ethereum
Node Architecture	The software architecture of an Ethereum node between the consensus and execution clients
Networks	Overview of the different Ethereum networks
Consensus Mechanisms	High level overview of how consensus works
Proof of Work	Covers how Ethereum consensus worked pre-Merge
Proof of Stake	Covers how Ethereum consensus currently works
Standards	This covers EIPs and how Ethereum upgrades itself as well as the token standards that have been implemented

Finish remaining Introduction to Ethereum Section on Base Camp (excluding the Guide to Base) (30 min)

To understand where Ethereum is headed, this graphic covers all the changes that have happened and are coming to the protocol

# index.md:

Home | Previous | Next

Smart Contracts and Solidity

<p><img src="./logo.svg"/></p>

Smart Contracts are coded contracts deployed onto a blockchain network. These contracts are immutable and transactions executed on a smart contract are irreversible. Understanding how smart contracts work is key to understanding how Ethereum and Base work. The most popular language for writing smart contracts is Solidity, which is a high-level, curly bracket, statically typed language. All of the guides linked below use Solidity as their language of choice, and the best reference for the language is the docs portal. Anytime you run into an issue with Solidity, this is the first place to look for answers.

Base Camp

(2 hours) provides a good introduction to Solidity development with specific modules for token development and building with Hardhat, a popular framework for Ethereum development. You should complete the full set of tutorials before moving forward.

The Ethereum foundation provides a guide to the ETH Stack. Most of the material on specific tools are covered in the Base Camp tutorials, so the most important articles from this are listed below:

Title	Description
Intro to the Stack	High level overview of the Ethereum Stack from the EVM up to End-User Dapps
Smart Contracts	Overview of how Smart Contracts work
Smart Contracts Languages	Covers the programming languages used to develop Smart Contracts on the EVM
Smart Contract Anatomy	Describes the technical components within a smart contract including storage, functions, events, etc
Smart Contract Libraries	Explains what Smart Contract libraries are and why they are used
Testing Smart Contracts	The why and how with regards to testing a Smart Contract
Decentralized Storage	How storage works in a decentralized blockchain network

There are several long tutorials/bootcamps for learning how to develop with Solidity and Ethereum beyond Base Camp. Alchemy University is a free 7 week full bootcamp that starts with building on data structures through various Solidity development modules.

Devpill.me is a fantastic resource for learning all aspects of development on blockchains. It covers not only Solidity/Smart Contract development but also Frontend, Backend and more roles within the blockchain ecosystem.

For a fun learning experience of building a game, CryptoZombies teaches how to build blockchain dapps via simple games.

Before we move on to learning about Base and Optimistic Rollups, there are a couple of smart contract patterns you should be familiar with. These are frequently used for upgradeability onchain. Given that deployed contracts are immutable, several patterns have been developed to allow for versioning between different deployed versions of a smart contract.

Pattern	EIP	Examples
Upgradeable Proxy	EIP-1967	Solidity By Example
Diamond Pattern	EIP-2535	Medium Article on Pattern

Most if not all smart contract patterns are added to Ethereum through the EIP process as detailed on their doc site under Standards. Ethereum is always under development, and new proposals add great features to the blockchain. Because Base is built on top of Ethereum, these proposals add features that can be asynchronously integrated onto Base as well. To learn more about how EIPs propagate to layer 2s, read this article by

Going forward many of the articles will link to specific smart contracts and EIPs. If you’re curious on how that works in code, Solidity By Example has many reference examples, and we’ll be calling out specific ones along with articles explaining how they work.

# README.md:

Native Bridge Examples

This repository collects a couple examples of how to call the smart contracts to bridge native ETH and ERC20 tokens to Base

Environment Setup

Clone this repo

```bash
git clone https://github.com/base-org/guides
```

Change to the directory

```
bash
cd guides/bridge/native
```

Install the dependencies:

```
bash
npm install
```

UI

This is a simple example using OP SDK to bridge native ETH.

Demo

Code

Hardhat Scripts

Copy secrets.json.sample to secrets.json

```
cp secrets.json.sample secrets.json
```

Add in your test wallet mnemonic and an alchemy key

If you are new to alchemy, a quickstart is available here

To verify the values are correct, check the balance with.

```
npx hardhat balance
```

Deposits

You can bridge ETH.

```
npx hardhat bridge --amount 0.1
```

You can bridge any supported ERC20 token.

WETH on goerli: 0xB4FBF271143F4FBf7B91A5ded31805e42b2208d6

WETH on base-goerli: 0x4200000000000000000000000000000000000006

If you need WETH, you can wrap easily on uniswap

Cross-chain token addresses can be found in the Optimism Token List

You bridge a token with

```
npx hardhat bridgeToken --amount 0.01 --l1token
0xB4FBF271143F4FBf7B91A5ded31805e42b2208d6 --l2token
0x4200000000000000000000000000000000000006
```

Withdrawals

You can fetch existing withdrawals with

```
npx hardhat fetchWithdrawals
```

Withdrawals will have 4 booleans to indicate it's lifecycle. isReadyToProve, isProven, isReadyToFinalize, and isFinalized

To initiate a native token withdrawal, start with

```
npx hardhat withdraw --amount 0.01
```

The withdrawal will enter a proposing onchain state. Once the withdrawal is proposed and messaged between layer one, the withdrawal can be verified.

```
npx hardhat proveWithdrawal --tx {your transaction hash from above}
```

Once the transaction is proven, the withdrawal has a holding period until it can be finalized. Currently in testnet, this is 12 seconds. Mainnet is 7 days.

Lastly, you can finalize the transaction with

```
npx hardhat finalizeWithdrawal --tx {your transaction hash from above}
```

The same flow can be used for ERC20 tokens. Make sure that the token is supported by the bridge by checking the l2 base, base-goerli addresses and l1 ethereum, goerli addresses in the optimism token list

```
npx hardhat withdrawToken --amount 0.01 --token
0x4200000000000000000000000000000000000006
```

Add your token to Base

If you already have an ERC20 on Ethereum and want to bridge to Base, a guide is available here

# README.md:

Introduction

EIP-6551 enhances the features of NFTs (ERC-721 tokens) by giving each token its own smart contract account, also known as token-bound accounts (TBA). These smart contract accounts equip NFTs with the ability to operate like traditional blockchain wallets, allowing state changes and features such as holding ERC-20 tokens, other NFTs, and ether. Programmable accounts open up a myriad of use cases, limited only by the developer's imagination and the protocol's constraints.

Importantly, EIP-6551 is backwards compatible and doesn't mandate custom logic within the NFT smart contract. This compatibility allows NFT projects that were created before the introduction of EIP-6551 to adopt smart contract accounts via a permissionless registry.

Use cases for token bound accounts:

- Gaming - A character, represented as an NFT, can possess its own inventory (like weapons, gold, etc.) using a smart wallet
- Investing - Utilize NFTs and their smart contract accounts to organize assets and sell entire investment portfolios in a single transaction
- Social (DAOs) - NFTs representing DAO membership can now maintain their own transaction history

Create an NFT with a Token Bound Account

This guide demonstrates how to create and interact with three smart contracts:

1. An ERC-721 smart contract that serves as our NFT.
2. A registry contract that deploys smart contract account associated with a given ERC-721 and computes the token bound account address.
3. A token bound account contract that can send and receive ether and other ERC-20 tokens.

Tools

We will use a few tools to write and deploy our smart contracts:

Tool	Description
Hardhat	Helps deploy and interact with smart contracts

Node.js	Developer environment
OpenZeppelin	An open library for building secure smart contracts
Coinbase Wallet	Non-custodial wallet for creating accounts and interacting with the blockchain

Getting started

We will be deploying these smart contracts on the Base Goerli testnet allowing us to see transaction information on BaseScan.

Using a testnet requires us to obtain testnet funds from a faucet to deploy and interact with our smart contracts on that network.

Configure Coinbase Wallet for testnet interactions

Warning: This demo requires the use of private keys in order to effectively interact with the blockchain. For this demo, use newly created wallets as a safety measure.

Depending on system settings your wallet may look different

Open the Coinbase Wallet browser extension then click on the Settings tab

!Settings tab

Select "Developer Settings"

!Developer settings

Toggle "Testnets" on

!Enable testnets

Request Testnet Funds

Click on the Settings tab

!Settings tab

Select "Networks"

!Networks tab

Select "Testnets" tab

!Testnets

Click on the water (💧) icon

!Request funds

Request testnet funds

!Request funds

Alternatively, use the base faucet

Environment Setup

Clone this repo

bash
```
git clone https://github.com/base-org/guides.git
```

Change into the directory

bash
```
cd base-guides/6551
```

Initiate a node project and install dependencies:

The dependencies in this project are as follows:

Dependencies	Description
hardhat	Helps deploy and interact with smart contracts
hardhat-toolbox	Bundles all the commonly used packages and Hardhat plugins we recommend to start developing with Hardhat.
dot-env	Dotenv is a zero-dependency module that loads environment variables from a .env file into process.env
openzeppelin/contracts	A library for secure smart contract development.

Install them by running:

bash
```
npm install
```

Wallet setup

Assuming you have set up your wallet (and safely stored away your seed phrase) and have some funds (testnet or mainnet), obtain the addresses and private keys needed for the demo.

Copy your private key

Do not share your private key with anyone.

Click on the Settings tab

!Settings tab

Select "Developer Settings"

!Developer settings

Click on "Show private key"

!Show private key

Enter password

!Enter password

Read disclaimer to copy address

!accept terms

Type the following command into your terminal application:

```
bash
export WALLETKEY=
```

Paste the private key after export WALLETKEY= within your terminal

Your terminal should look like this

```
bash
export
WALLETKEY=0xde9be858da4a475276426320d5e9262ecfc3ba460bfac56360bfa6c4c28b4
ee0
```

Press enter/return on your keyboard to save the value

Verify your private key has been saved:

```
bash
echo $WALLETKEY
```

Sample output:

```
bash
0xde9be858da4a475276426320d5e9262ecfc3ba460bfac56360bfa6c4c28b4ee0
```

Obtain details of another account

Note: Each account will need funds in order to deploy contracts and interact with Base.

From the assets tab, click on the current address/account

!Assets tab

Select another wallet/account

If you do not have an additional account click the "Add & manage wallets" button to create a new account.

!Select another account

Copy the address of the newly selected account

!Copy address

Type the following command into your terminal application

Your terminal should look like this:

```
bash
export WALLET2ADDR=
```

Paste the address as an environment variable within your terminal

```
bash
export WALLET2ADDR=0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
```

Press enter/return on your keyboard to save the value

Verify the account address has been saved:

```
bash
echo $WALLET2ADDR
```

Sample output:

```
bash
0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
```

Save the private key of the currently selected account

Each account has its own private key.

This wallet will be the original owner of the NFT minted later in this tutorial. You will need the private key of the currently selected wallet in order to transfer the ownership of the NFT.

Refer to the previous steps for obtaining and copying your private keys and save them as an environment variable WALLET2KEY using your terminal.

Example:

bash
```
export
WALLET2KEY=0x5de4111afa1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab
365a
```

Verify your private key has been saved:

bash
```
echo $WALLET2KEY
```

Sample output:

bash
```
0x5de4111afa1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab365a
```

Deploy contracts on the Base testnet

The following script will deploy three smart contracts and save their deployment information in a file called deploymentData.json.

Once you become comfortable deploying contracts on a testnet, you may deploy contracts on the Base mainnet by replacing --network base-goerli with --network base-mainnet .

```
npx hardhat run scripts/01deploycontracts.js --network base-goerli
```

Sample output:

bash
```
ERC-721 Contract deployed at: 0xECD4b1C01a0aF70C7Ee985db389E294D01DffEC0
Saved deployment data to a new file: deploymentData.json
Deployed registry contract at: 0x58B2EAe6f05abf9C1e1566AD9307C67B41627A1e
```

bash
```
[
  {
  NftContract: {
  address: '0xECD4b1C01a0aF70C7Ee985db389E294D01DffEC0',
  deployer: '0xB6d00D83158feE6695C72ff9c5E915478A465724',
  deploymentHash:
'0xedacc11c5268b155a9a6918b5e1cc19031343f41519f596eb264ef6ca3feaeb4'
  }
  },
  {
  ERC6551Registry: {
  address: '0x58B2EAe6f05abf9C1e1566AD9307C67B41627A1e',
  deployer: '0xB6d00D83158feE6695C72ff9c5E915478A465724',
  deploymentHash:
'0xcf45022f97e682eccc87c903c6eff38ba2c080aeb69c69da4662914aacf4f481'
  }
  }
]
```

bash

```
  Deployment data saved to: deploymentData.json
  Deploying Token Bound Account
  Token bound account deployed at:
0xdAcaEDF79Fa33405446F2B9Fbf820Cef84507f22
```

bash

```
[
  {
    NftContract: {
    address: '0xECD4b1C01a0aF70C7Ee985db389E294D01DffEC0',
    deployer: '0xB6d00D83158feE6695C72ff9c5E915478A465724',
    deploymentHash:
'0xedacc11c5268b155a9a6918b5e1cc19031343f41519f596eb264ef6ca3feaeb4'
    }
  },
  {
    ERC6551Registry: {
    address: '0x58B2EAe6f05abf9C1e1566AD9307C67B41627A1e',
    deployer: '0xB6d00D83158feE6695C72ff9c5E915478A465724',
    deploymentHash:
'0xcf45022f97e682eccc87c903c6eff38ba2c080aeb69c69da4662914aacf4f481'
    }
  },
    {
    ERC6551Account: {
    address: '0xdAcaEDF79Fa33405446F2B9Fbf820Cef84507f22',
    deployer: '0xB6d00D83158feE6695C72ff9c5E915478A465724',
    deploymentHash:
'0xbfe9ab08951e07660c2d3a6e9b16e6e959a174530f1e28aba9c98a19c381f586'
    }
  }
]
```

bash

```
Deployment data saved to: deploymentData.json
```

This script will mint an NFT and assign its ownership to the WALLET2ADDR account

```
bash
npx hardhat run scripts/02mintnft.js --network --network base-goerli
```

Sample output:

```
bash
0xECD4b1C01a0aF70C7Ee985db389E294D01DffEC0
Minting NFT...
TokenId 0 is owned by address: 0x9eEd71442F60440b39Def927047e5823c0b208D4
```

This script will create and compute the address for a smart contract wallet (token bound account)

```
bash
npx hardhat run scripts/03createaccount.js --network base-goerli
```

Sample output:

```
Computed Address: 0xA5153E5D9A384e519fEa64D228797edb4a448d45
```

This script will send funds from WALLETKEY to the token bound account and transfer ownership of the NFT from WALLET2ADDR to WALLETKEY.

```
bash
npx hardhat run scripts/04accountinteraction.js --network base-goerli
```

Sample output:

```
Current owner of tokenId 0 is 0x9eEd71442F60440b39Def927047e5823c0b208D4
Token account has 0 ETH
New owner of tokenId 0 is 0xB6d00D83158feE6695C72ff9c5E915478A465724
Token account has 12500000000000000 ETH
```

Congrats!

Clear the private keys from your environment variables by running this command:

```
bash
export WALLETKEY=nil
export WALLET2KEY=nil
```