



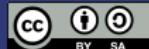
# Intro to ConsenSys and Blockchain Development

*A primer*

By Marc Lijour

October 13, 2018

One of the largest blockchain companies in the world with 1,000 employees in 30 countries



# Who am I?

<https://www.linkedin.com/in/marclijour/>



**COLLIDER-X**



**CONSENSYS**



Access these slides

<https://bit.ly/2qNBiHw>

or find by date:

<https://github.com/marclijour/presentations>



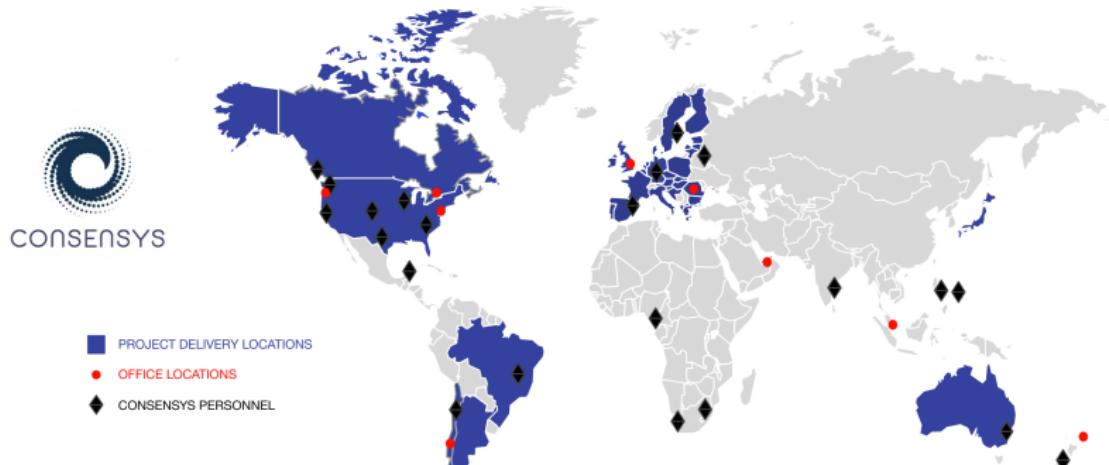
# Table of Contents

- 1 What is ConsenSys?
- 2 Introduction to Ethereum
- 3 Setting a Development Machine
- 4 ConsenSys Stack Overview
- 5 Hands-on Transactions & Smart Contracts
- 6 Developing with Truffle
- 7 Private blockchain experiment



# The largest Blockchain Cie in the world

We are 1000+ blockchain experts, entrepreneurs, computer scientists, designers, engineers, consultants, and business leaders across 6 continents



# Transforming Industries



ConsenSys partnered with a consortium of 15 key industry leaders and leading bank institutions to create a network-based commodities trading platform.



# Transforming Industries



**UNIONBANK**

ConsenSys partnered with UnionBank to develop a closed-loop crypto-cash solution in the Philippines to connecting rural banks to the main financial infrastructure.

**KALEIDO**

A ConsenSys Business

Try it on aws marketplace



# Transforming Industries



mcCarthy  
tetraul

ConsenSys has partnered with McCarthy-Tétrault to automate key aspects of the lending process by leveraging smart contract powered loan agreements on the Ethereum blockchain.

## In this demo....



- OpenLaw's powerful markup language
- Digital, blockchain-based signatures
- Smart contract execution to a decentralized app ("dApp")



# Fair Token Launches

## The Brooklyn Project Framework

### CIVIL Transparency Scorecard

as of 18 September 2018



#### Transparency Grading Scale

- 1 – 2 **Red flag** Non-existent or conflicting disclosures.
- 3 – 4 **Poor** Failure to reveal useful information.
- 5 – 6 **Lacking** Partially complete or unclear disclosures.
- 7 – 8 **Good** Standards met, but room for improvement.
- 9 – 10 **Very good** Complete, clear, consistent disclosures.

<https://framework.thebkp.com/project/CVL/document/2/version/3>

TOKEN FOUNDRY



<https://tokenfoundry.com/projects/civil>



# Table of Contents

- 1 What is ConsenSys?
- 2 Introduction to Ethereum
- 3 Setting a Development Machine
- 4 ConsenSys Stack Overview
- 5 Hands-on Transactions & Smart Contracts
- 6 Developing with Truffle
- 7 Private blockchain experiment



# Ethereum

Ethereum is a **decentralized platform that runs smart contracts**: applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference.

— <https://ethereum.org>



# A short history of Ethereum

## Key Milestones:

- (late 2013) Vitalik Buterin describes Ethereum in a paper



# A short history of Ethereum

## Key Milestones:

- (late 2013) Vitalik Buterin describes Ethereum in a paper
- (Summer 2014) Ethereum raises more than \$14 million in pre-sale



# A short history of Ethereum

## Key Milestones:

- (late 2013) Vitalik Buterin describes Ethereum in a paper
- (Summer 2014) Ethereum raises more than \$14 million in pre-sale
- (July 30, 2015) Launch of Frontier, initial (beta) version of Ethereum



# A short history of Ethereum

## Key Milestones:

- (late 2013) Vitalik Buterin describes Ethereum in a paper
- (Summer 2014) Ethereum raises more than \$14 million in pre-sale
- (July 30, 2015) Launch of Frontier, initial (beta) version of Ethereum
- (March 14, 2016) Launch of Homestead, first production release



# A short history of Ethereum

## Key Milestones:

- (late 2013) Vitalik Buterin describes Ethereum in a paper
- (Summer 2014) Ethereum raises more than \$14 million in pre-sale
- (July 30, 2015) Launch of Frontier, initial (beta) version of Ethereum
- (March 14, 2016) Launch of Homestead, first production release
- (Spring 2016) The DAO



# A short history of Ethereum

## Key Milestones:

- (late 2013) Vitalik Buterin describes Ethereum in a paper
- (Summer 2014) Ethereum raises more than \$14 million in pre-sale
- (July 30, 2015) Launch of Frontier, initial (beta) version of Ethereum
- (March 14, 2016) Launch of Homestead, first production release
- (Spring 2016) The DAO
- (July 2, 2016) ETH – ETC split



# A short history of Ethereum

## Key Milestones:

- (late 2013) Vitalik Buterin describes Ethereum in a paper
- (Summer 2014) Ethereum raises more than \$14 million in pre-sale
- (July 30, 2015) Launch of Frontier, initial (beta) version of Ethereum
- (March 14, 2016) Launch of Homestead, first production release
- (Spring 2016) The DAO
- (July 2, 2016) ETH – ETC split
- (October 16, 2017) Launch of Metropolis (vByzantium) –version 3



# A short history of Ethereum

## Key Milestones:

- (late 2013) Vitalik Buterin describes Ethereum in a paper
- (Summer 2014) Ethereum raises more than \$14 million in pre-sale
- (July 30, 2015) Launch of Frontier, initial (beta) version of Ethereum
- (March 14, 2016) Launch of Homestead, first production release
- (Spring 2016) The DAO
- (July 2, 2016) ETH – ETC split
- (October 16, 2017) Launch of Metropolis (vByzantium) –version 3
- (2017) ETH goes from \$7 to more than \$700 (100x increase)

*Check the nice infographic (Invezz, 2017).*

## More information:

- a “prehistory” of the Ethereum protocol (Buterin, 2017).
- the official *Ethereum White Paper*.



# Decentralization

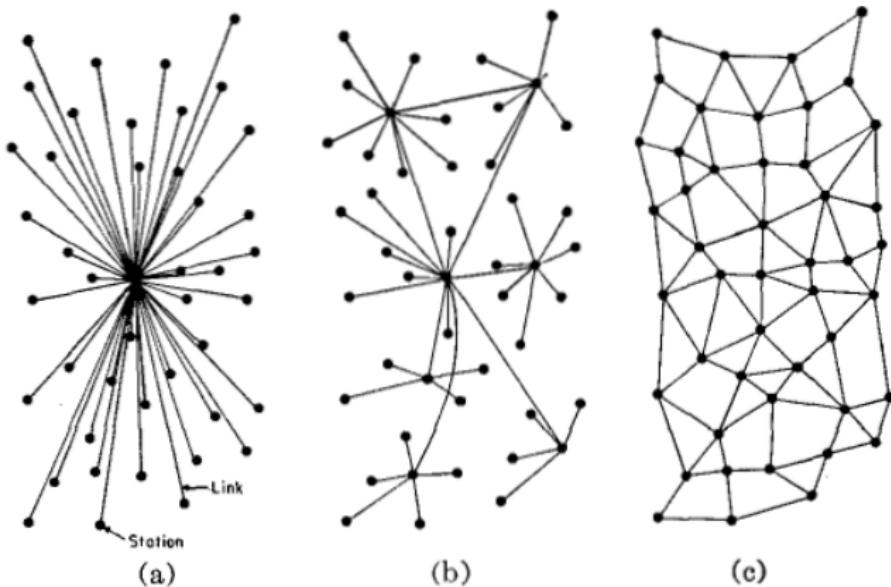


Fig. 1—(a) Centralized. (b) Decentralized. (c) Distributed networks.



# Client Types

- Full node



# Client Types

- Full node
- Light node



# Client Types

- Full node
- Light node
- Something in between (e.g. “fast” for geth)



# Disk Space

Full Archive Ethereum node

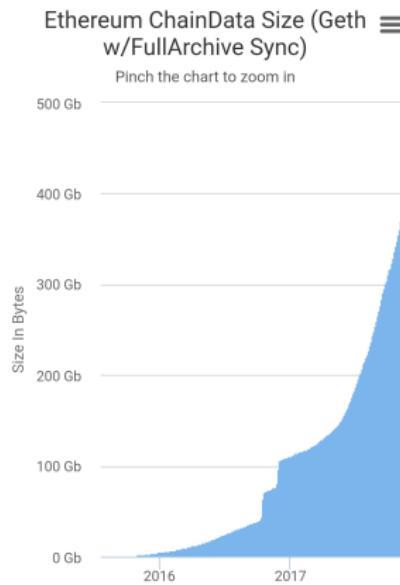


Figure: Miners need a lot of space (Reddit, 2017)



# Disk Space

Ethereum vs. Bitcoin

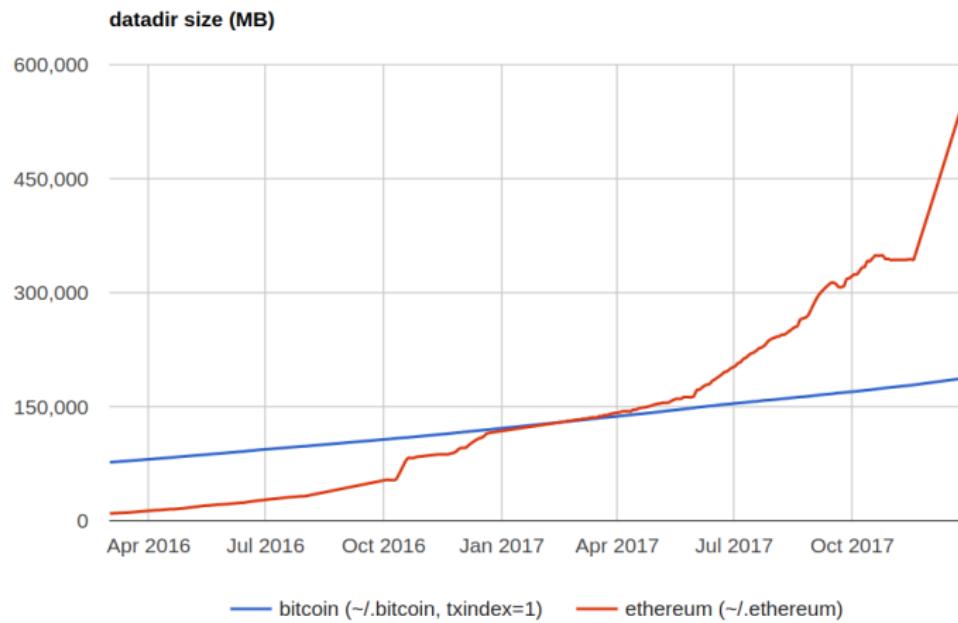


Figure: Disk space used by Geth (fast) vs. Bitcoin (Daniel, 2017)



# Disk Space

With Geth --syncmode fast (default mode)

This mode initializes a ~20 GB database, then turns in full node.

The GETH client has 3 Blockchain sync modes (fast, full or light). The 'FAST' sync was used to produce the data chart below using Geth v1.6.7 stable.

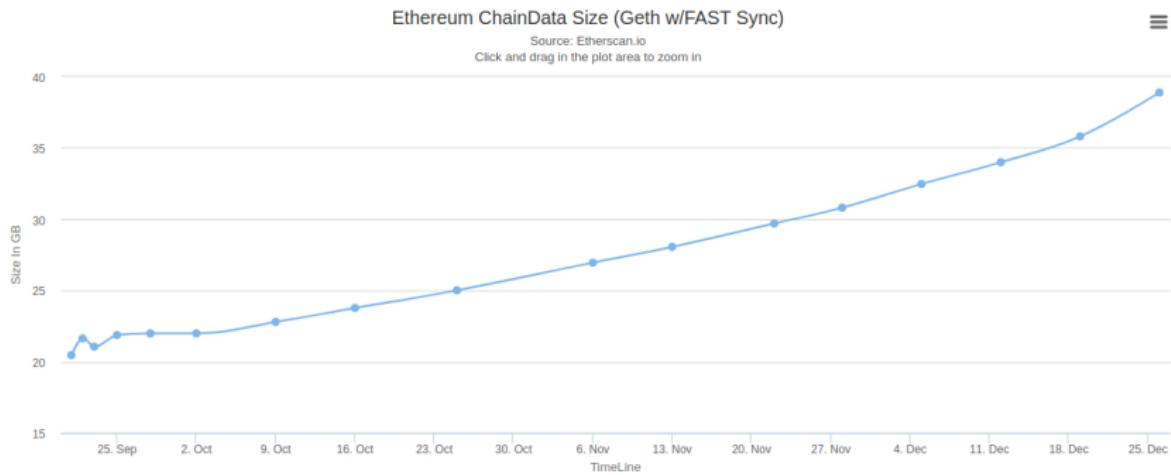


Figure: Disk space used by Geth (in fast mode) (Etherscan, 2017)



# Disk Space

Parity allows for continuous state trie pruning

In green, the configuration running as *full node*.  
A light client can fit in ~5 MB.

ID	Pruning Mode	Database Configuration	Block Verification	Available Blocks	Available States	Chaindata Size	Parity CLI Flags to use this configuration
0Archive	+Fat +Trace	Full	All	All		385.000 GB	-pruning archive --tracing on --fat-db on
1Archive	+Trace	Full	All	All		334.000 GB	-pruning archive --tracing on
2Archive		Full	All	All		326.000 GB	-pruning archive
3Fast	+Fat +Trace	Full	All	Recent		37.000 GB	--tracing on --fat-db on
4Fast	+Trace	Full	All	Recent		34.000 GB	--tracing on
5Fast		Full	All	Recent		26.000 GB	-no-warp
6Fast	+Warp	Ancient-PoW-Only	All	Recent		25.000 GB	
7Fast	+Warp -Ancient	No-Ancient	Recent	Recent		5.300 GB	-no-ancient-blocks
8Light		Headers-Only	None	None		0.005 GB	light

Figure: Disk space used by Parity (Afri, 2017)

# Metamask



METAMASK

Brings Ethereum to your browser

[GET CHROME EXTENSION ▶](#)

Chrome Firefox Opera

OR

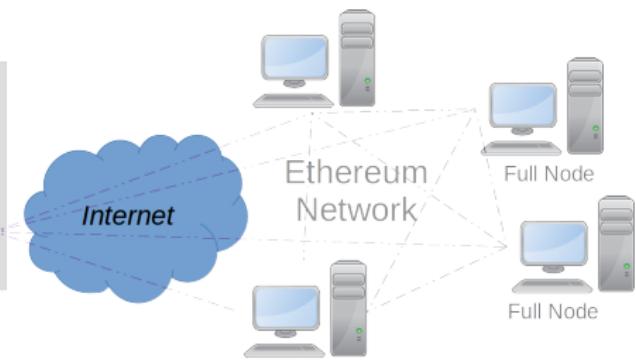
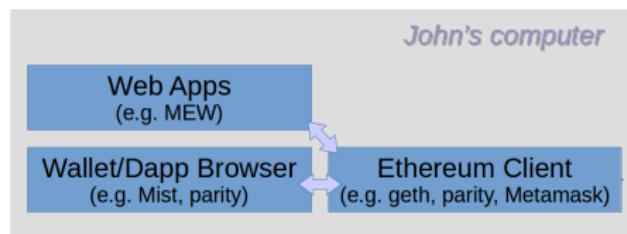
[GET BRAVE BROWSER ▶](#)

<https://metamask.io>



# Practical Applications

for personal or business use



# Table of Contents

- 1 What is ConsenSys?
- 2 Introduction to Ethereum
- 3 Setting a Development Machine
- 4 ConsenSys Stack Overview
- 5 Hands-on Transactions & Smart Contracts
- 6 Developing with Truffle
- 7 Private blockchain experiment



# Sizing up

- 1) Ethereum Node, Wallet, historical data, Smart Contracts, and Dapps:
  - Linux machine (Ubuntu 16.04 / Linux Mint 18.x –until April 2021)
  - Parity (or Geth)
  - A Solidity compiler
- 2) Developer light setup: (works on ChromeOS)
  - Chrome browser (or Chromium) –any OS
  - Metamask Extension
  - [Remix](#) IDE
- 3) Developer Pro setup:
  - truffle



# Parity

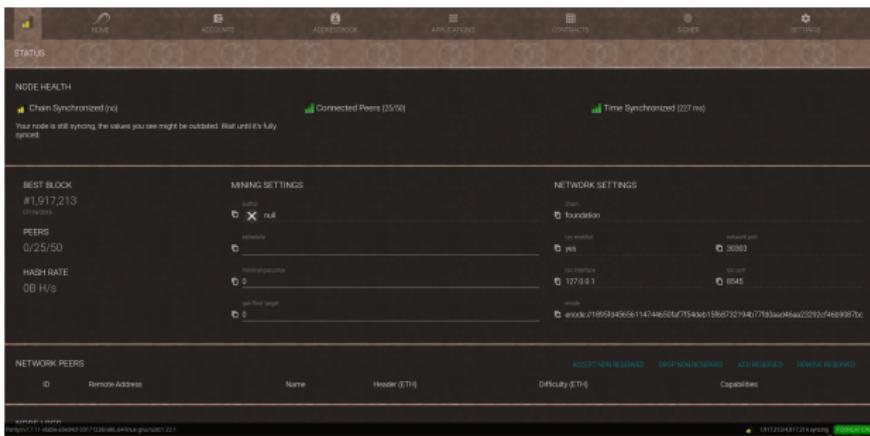


Figure: The **Parity** client syncing

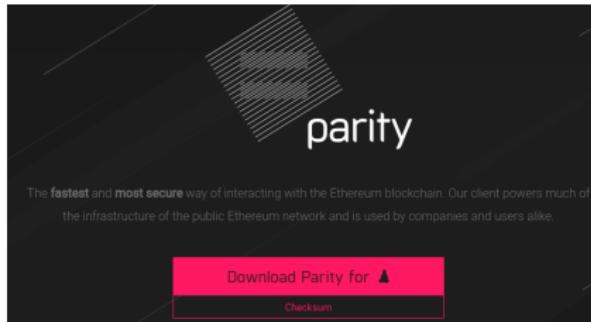
- Typical Account Management, multi-sig, hardware support
- Access Dapps directly (e.g. app to create an ERC-20 token)
- Code editor and Solidity compiler for smart contracts
- Fast and reliable (written in Rust)
- Most OS, Docker images; and compliant with JSON-RPC API



## Installing Parity



# Installing Parity



<https://www.youtube.com/watch?v=WNT2O6xyDmM> (Windows-based, 16 min)

- ① Go to <https://www.parity.io>
- ② Download the relevant binaries, e.g. on Linux:
- ③ Check the checksum: `$ md5sum parity_1.7.11_amd64.deb`
- ④ Install: `$ sudo dpkg -i parity_1.7.11_amd64.deb`
- ⑤ Check the version: `$ parity -v`



# Run Parity on the Kovan Testnet

```
$ parity --light --testnet
2017-12-28 23:38:25 Starting Parity/v1.7.11-stable-a5ed4cf-20171228/x86_64-linux-gnu/rustc1.22.1
2017-12-28 23:38:25 Keys path /home/marc/.local/share/io.parity.ethereum/keys/Kovan
2017-12-28 23:38:25 DB path /home/marc/.local/share/io.parity.ethereum/chains/kovan/db/9bf388941c25ea98
2017-12-28 23:38:25 Path to dapps /home/marc/.local/share/io.parity.ethereum/dapps
2017-12-28 23:38:25 Running in experimental Light Client mode.
...
...
```

Then go to <http://localhost:8180> (or <http://web3.site> if online), and follow the instructions.

- After reading the legal terms and conditions, you can create your first account.
- Click on the top left-most logo (yellow bars) to see the status of your node.
- **It may take days to sync!**



# Try running your first Dapp

Follow the tutorial at

<https://wiki.parity.io/Deploying-Dapps-to-Parity-Wallet> (using chevdor's dapp generator and yeoman)

On Linux Ubuntu, make sure you have npm, and make a soft link to node before running init.sh:

```
$ sudo apt install npm  
$ sudo ln -s /usr/bin/nodejs /usr/bin/node  
$ ./init.sh
```



## Installing Geth



# Installing Geth

Instructions (all OSes) at

<https://github.com/ethereum/go-ethereum/wiki/Building-Ethereum>.

Ubuntu/Mint: <https://github.com/ethereum/go-ethereum/wiki/Installation-Instructions-for-Ubuntu>

```
$ sudo apt-get install software-properties-common  
$ sudo add-apt-repository -y ppa:ethereum/ethereum  
$ sudo apt-get update
```

Run the first line to install the full suite (geth, bootnode, evm, disasm, rlpdump, ethtest), or the second line for geth only:

```
$ sudo apt-get install ethereum  
$ sudo apt-get install geth
```

Create a new account, and you should be ready to run geth:

```
$ geth account new  
$ geth
```



# Installing a Solidity Compiler

Provided the previous steps were completed:

```
$ sudo apt-get install solc  
$ which solc
```

And in geth, to let it know where solc can be found:

```
$ admin.setSolc("/usr/bin/solc")
```

Now test the code by following the instructions at

<https://github.com/ethereum/go-ethereum/wiki/Contract-Tutorial>



# Code Editor



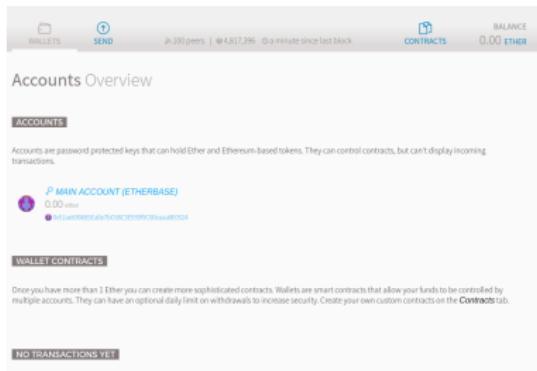
- Vim
- Vim Solidity
- Vim Syntastic



# And you still need a wallet

## Options:

- Mist Browser (beta) (featured on the right, see also the recent security warning re. Chromium)
- MyEtherWallet (MEW) supports advanced features including hardware wallets



Mist Browser (beta)  
<https://wallet.ethereum.org>  
Try on Chrome vs Firefox



# Table of Contents

- 1 What is ConsenSys?
- 2 Introduction to Ethereum
- 3 Setting a Development Machine
- 4 ConsenSys Stack Overview
- 5 Hands-on Transactions & Smart Contracts
- 6 Developing with Truffle
- 7 Private blockchain experiment

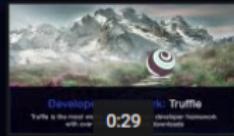


# ConsenSys Dev Tools in Numbers

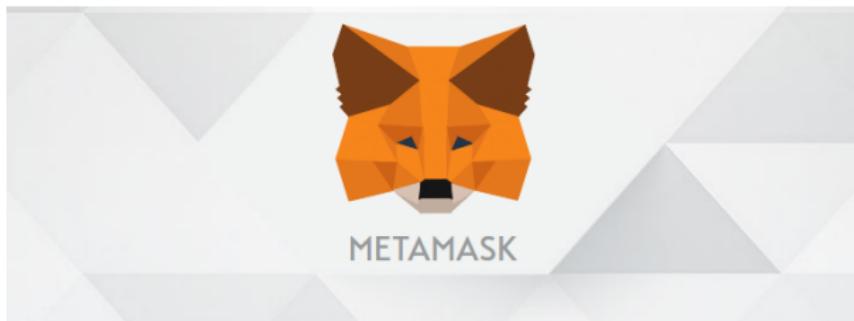
<https://www.youtube.com/watch?v=WKwR51ANy9E>

## ConsenSys+Ethereum

ConsenSys develops tools and infrastructure



# Metamask



Brings Ethereum to your browser

[GET CHROME EXTENSION ►](#)

Chrome Firefox Opera

OR

[GET BRAVE BROWSER ►](#)

<https://metamask.io>



# Truffle Framework

<http://truffleframework.com>

## YOUR ETHEREUM SWISS ARMY KNIFE

Truffle is the most popular development framework for Ethereum with a mission to make your life a whole lot easier.

★ Star 4,734

Fork 594

[gitter](#) [join chat](#)

INSTALL VIA NPM

```
$ npm install -g truffle
```

*Requires NodeJS 5.0+. Works on Linux, macOS, or Windows.*

[DOCUMENTATION](#)

[TUTORIALS](#)

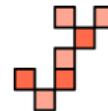
Don't know where to start? Get yourself a [Truffle Box!](#)





## BLOCKCHAIN BASED

We eliminate the need to install, configure, and maintain costly Ethereum infrastructure.



## RELIABLE AND SCALABLE

Our Ferryman™ middleware improves reliability and helps us scale quickly to meet your demand.



## DISTRIBUTED STORAGE

Access IPFS seamlessly without the hassle of managing the infrastructure.

## POWERFUL AND SECURE

5B+

Requests Per Day

1.6PB

Data Transferred Per Month

9000+

Developers and DApps Served



# Mythril

<https://github.com/ConsenSys/mythril>

Mythril is a security analysis tool for smart contracts.

It comes as a Python package that requires a solidity compiler and a C++ compiler.

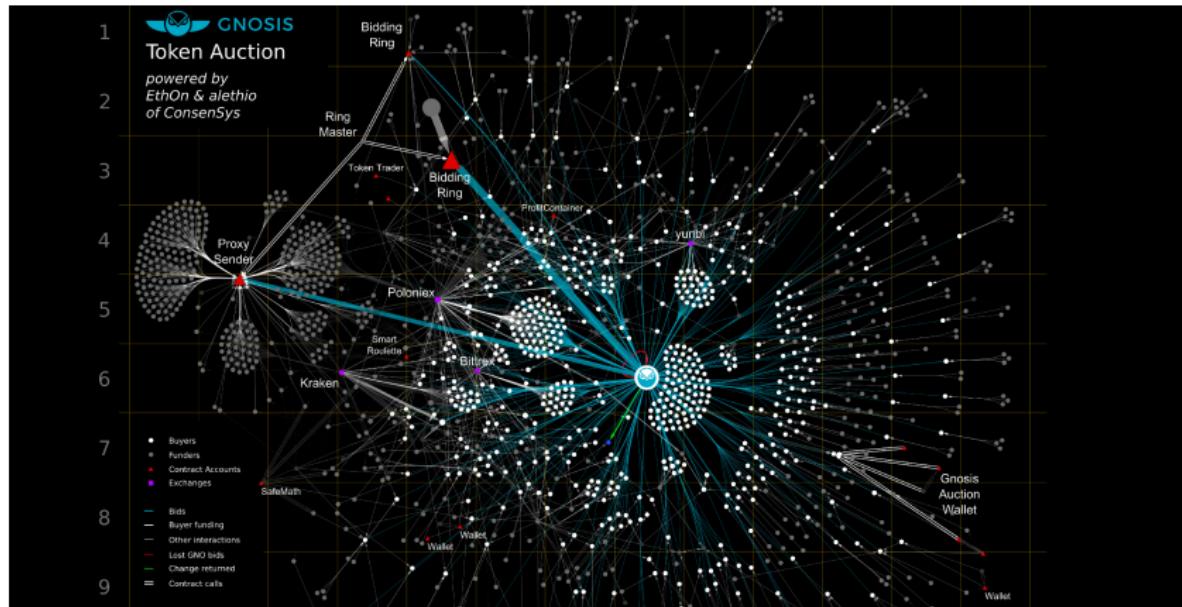
```
$ sudo apt install libssl-dev  
$ sudo apt install gcc g++  
$ sudo add-apt-repository ppa:ethereum/ethereum  
$ sudo apt install solc  
$ sudo pip3 install mythril  
$ myth -x contracts/higherbidder.sol
```

See also <https://hackernoon.com/introducing-mythril-a-framework-for-bug-hunting-on-the-ethereum-blockchain-9dc5588>



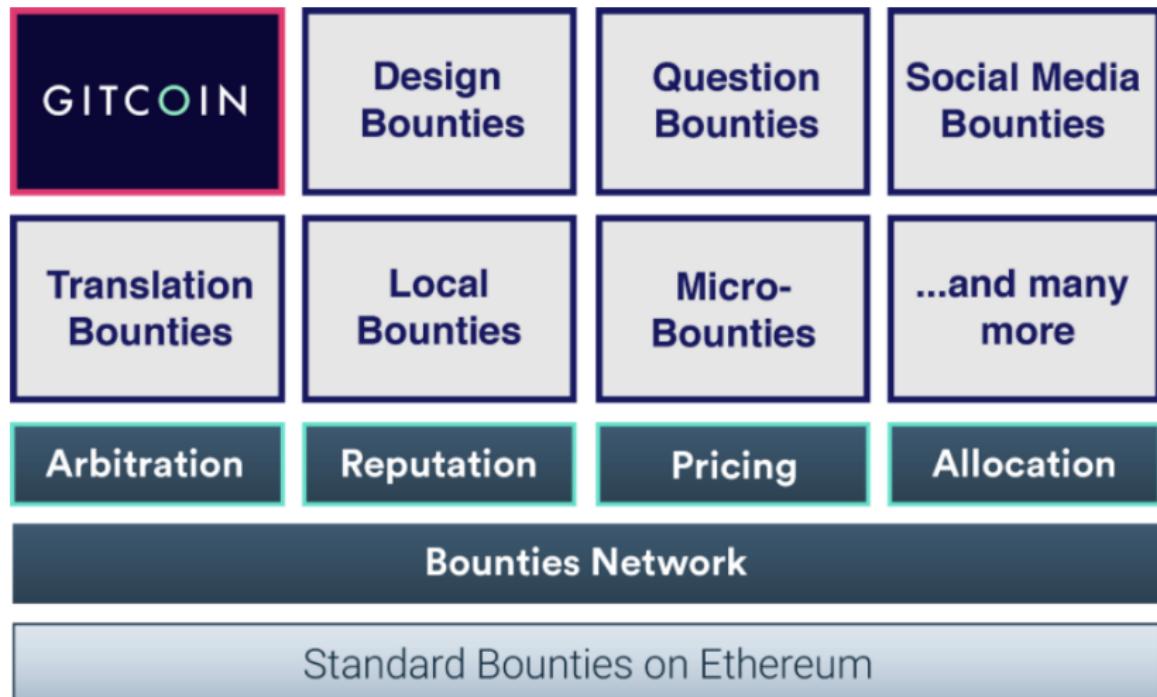
# Big Data and Analytics on Ethereum

<https://aleth.io>



# Getting paid for your work: The Bounties Network

<https://bounties.network>



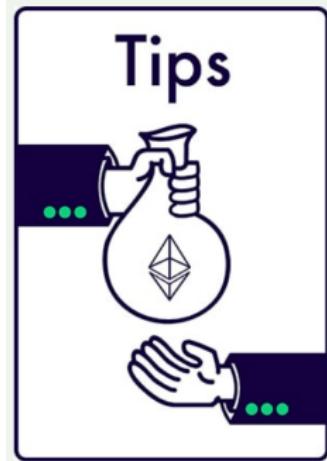
Gitcoin (depth-first) and Bounties Network (breadth-first) have integrated!



# Gitcoin

<https://gitcoin.co>

WHAT'S THE TLDR?



+



=



# Table of Contents

- 1 What is ConsenSys?
- 2 Introduction to Ethereum
- 3 Setting a Development Machine
- 4 ConsenSys Stack Overview
- 5 Hands-on Transactions & Smart Contracts
- 6 Developing with Truffle
- 7 Private blockchain experiment



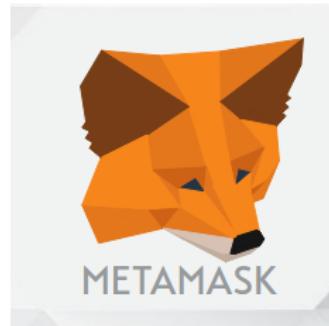
# Let's have some fun!



# Install MetaMask

Follow step by step:

- ① Install the [Chrome/Chromium extension](#)
- ② Watch the [intro on Youtube](#)
- ③ Create an account
- ④ Switch to the Ropsten Testnet (top-left in MetaMask)
- ⑤ Fill your account with Ether from  
<https://faucet.metamask.io>



<https://metamask.io>



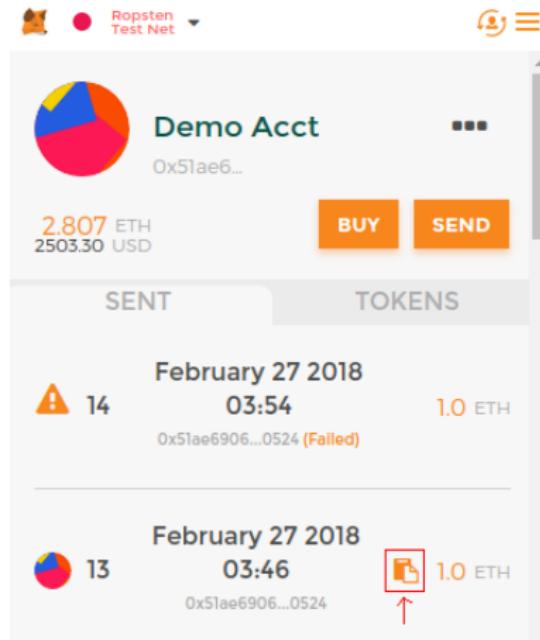
# Try sending ETH to yourself with Metamask

- ① Make sure you're on Ropsten, with some ETH from the faucet
- ② Click on "Send" and fill:
  - Account: paste your own address (same account)
  - Amount: 1 ETH
  - Transaction data: convert some text in HEX format with  
<https://www.asciitohex.com>, remove all spaces and write it with a 0x prefix (e.g. 0x497427732074696d6520746f2072756e)
- ③ Click on "Next"
- ④ Use a gas price > 30 (the higher the faster)
- ⑤ Confirm the transaction



# Check the transaction on Etherscan

Copy the transaction #



# Check the transaction on Etherscan

and click on "Convert to Ascii"

The screenshot shows the Etherscan interface for the Ropsten Testnet. At the top, there's a navigation bar with links for HOME, BLOCKCHAIN (selected), ACCOUNT, TOKEN, CHART, and MISC. A search bar contains the transaction hash 0x023983a0a803e906879dab806d4ada22a7776445264c8f671f92ead92415d65b, with a 'GO' button. Below the header, the transaction details are displayed:

Transaction Information	
TxHash:	0x023983a0a803e906879dab806d4ada22a7776445264c8f671f92ead92415d65b
TxReceipt Status:	Success
Block Height:	2735788 (76 block confirmations)
TimeStamp:	23 mins ago (Feb-27-2018 08:47:28 AM +UTC)
From:	0x51ae690685ea0e7bc68c3e939f9c00aaaa8e0524
To:	0x51ae690685ea0e7bc68c3e939f9c00aaaa8e0524
Value:	1 Ether (\$0.00)
Gas Limit:	51000
Gas Used By Txn:	22088
Gas Price:	0.00000004 Ether (40 Gwei)
Actual Tx Cost/Fee:	0.00088352 Ether (\$0.000000)
Cumulative Gas Used:	58002
Nonce:	13
Input Data:	0x497427732074696d6520746f2072756e

At the bottom of the input data section is a button labeled "Convert To Ascii".

# A note about gas price

<https://ethgasstation.info>

ETH Gas Station

Estimates over last 1,500 blocks - Last update: Block 5164391

Change Currency ▾

**Std Cost for Transfer** \$0.056

**Gas Price Std (wei)** 3

**SafeLow Cost for Transfer** \$0.056

**Gas Price SafeLow (wei)** 3

**Median Wait (s)** 29

**Median Wait (blocks)** 2

**Gas-Time-Price Estimator:** For transactions sent at block: 5164391

Adjust confirmation time

Avg Time (min)	4.38
95% Time (min)	10.95
Gas Price (wei)*	3
Tx Fee (Flat)	\$0.056

Gas Used*	21000
Avg Time (blocks)	18.02
95% Time (blocks)	45.05
Tx Fee (ETH)	0.00005

**Real Time Gas Use: % Block Limit (last 10)**

Last Block: 5164391

**Transaction Count by Gas Price**

**Confirmation Time by Gas Price**

**Top 10 Miners by Blocks Mined:** Support for user transactions

**Recommended Gas Prices** (based on current network conditions)

Speed	Gas Price (wei)
SafeLow (<30m)	3
Standard (<5m)	3
Fast (<2m)	18

Note: Estimates not valid when multiple transactions are batched from the same address or for transactions sent to addresses with many (e.g. > 100) pending nonce conflicts.

**Misc Stats (Last 1,500 blocks)**

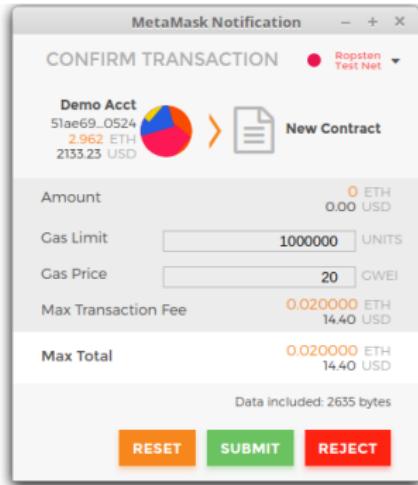


# Create your own (ERC-20) token

## Create Token

Create Token Contract with the following parameters.

100
Marc's Coin #2
8
MLD
<input type="button" value="Create Token"/>



- ① Use the Token Factory Dapp at <https://tokenfactory.surge.sh/#/factory>
- ② MetaMask will pop up (see picture above)
- ③ Submit the transaction (on the Ropsten Testnet)
- ④ Check your transaction on <https://ropsten.etherscan.io>



# Check your Smart Contract

A screenshot of a mobile application interface for checking smart contract status. At the top, there are two tabs: "SENT" (highlighted in grey) and "TOKENS". Below the tabs, the date and time are displayed as "December 29 2017 04:49". To the left of the date is a document icon with the number "2" next to it, indicating the count of published contracts. To the right of the date is an orange copy icon with "0 ETH" next to it, indicating the amount of Ether sent. Below the date, the text "Contract Published" is visible.

- ① Select the “Sent” tab
- ② Check the orange Copy icon (Tx Hash)
- ③ Click on “Contract Published”
- ④ That should bring you to Etherscan (see next page)

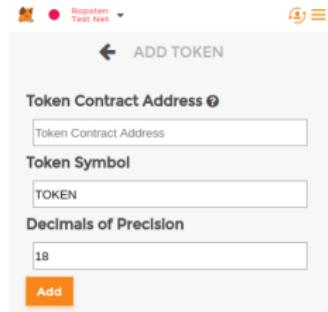
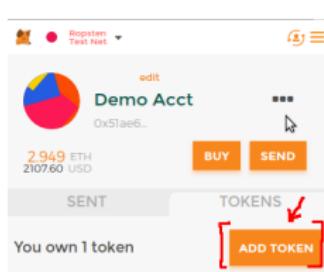


Verify the status of your transaction on Etherscan

Transaction Information: note the "To" line with your contract address



# Watch your Token



- ① Click on the “Add Token” button
- ② Wait for the next window (picture on the right)
- ③ Copy your contract address (from Etherscan)
- ④ Go back to your Token Factory tab, which should show an UI to interact with your contract or go to the URL:  
<https://tokenfactory.surge.sh/#/token/0x...> (replace 0x... by your contract address)
- ⑤ Move coins around
- ⑥ In MetaMask, click on your token to check the tx on Etherscan



# Too easy?

Let's code it in Solidity  
like the pros!



# Coding your first ERC-20 Smart Contract

```
pragma solidity ^0.4.0;
contract Ballot {
    struct Voter {
        uint weight;
        bool voted;
        uint8 vote;
        address delegate;
    }
    struct Proposal {
        uint voteCount;
    }
    address chairperson;
    mapping(address => Voter) voters;
    Proposal[] proposals;
    // Create a new ballot with _numProposals different proposals.
    function Ballot(uint8 _numProposals) public {
```

- ① Open the Remix IDE at <https://remix.ethereum.org>
- ② Close the ballot file
- ③ Create a new file named TokenRecipient.sol
- ④ Copy the code from <https://ethereum.org/token> (second white box, under “The Code”, starting with “pragma”)
- ⑤ Switch to the “Run” tab (top-right bar, after Compile)

Reference:

ERC-20 Token Standard



# Compiling Successfully

The screenshot shows the Truffle UI interface. On the left, there's a code editor window titled "browser/TokenRecipient.sol" containing Solidity code. On the right, there's a navigation bar with tabs for "Compile", "Run", "Settings", "Debugger", "Analysis", and "Support". Below the navigation bar, there's a button labeled "Start to compile" with an "Auto compile" checkbox. A dropdown menu shows "TokenERC20" selected. To the right of the dropdown are buttons for "Details" and "Publish on Swarm". A yellow box indicates "Static Analysis raised 4 warning(s) that require fixing". Below this are two green boxes: "TokenERC20" and "tokenRecipient".

```
1 pragma solidity ^0.4.16;
2
3 interface tokenRecipient { function receiveApproval(address _from, uint256 _value, i
4
5+ contract TokenERC20 {
6+     // Public variables of the token
7+     string public name;
8+     string public symbol;
9+     uint8 public decimals = 18;
10+    // 18 decimals is the strongly suggested default, avoid changing it
11+    uint256 public totalSupply;
12+
13+    // This creates an array with all balances
14+    mapping (address => uint256) public balances;
15+    mapping (address => mapping (address => uint256)) public allowances;
16+
17+    // This generates a public event on the blockchain that will notify clients
18+    event Transfer(address indexed from, address indexed to, uint256 value);
19+
20+    // This notifies clients about the amount burnt
21+    event Burn(address indexed from, uint256 value);
22+
23+}
```

- ① Two green boxes should show on the right
- ② TokenERC20 is the name of the contract (class)
- ③ tokenRecipient is the name of the interface
- ④ Switch to the “Run” tab (top right)



# Submitting the Smart Contract

The screenshot shows the Truffle UI interface. On the left is a code editor with the file 'browser/TokenRecipient.sol' containing Solidity code for a ERC20 token. The code defines a contract 'TokenERC20' with variables like name, symbol, and totalSupply, and functions for approval, transfer, and burn. On the right is a control panel with tabs for 'Compile', 'Run', 'Settings', 'Debugger', 'Analysis', and 'Support'. The 'Run' tab is active. It includes fields for 'Environment' (set to 'Injected Web3'), 'Account' (set to '0x51a...e0524'), 'Gas limit' (set to '3000000'), and 'Value' (set to '0 wei'). A dropdown menu shows 'TokenERC20'. Below it, there's a 'Create' button and a 'Load contract from Address' field. The status bar at the bottom shows '1 pending transactions' and '0 contract instances'.

```
pragma solidity ^0.4.16;

interface TokenRecipient {
    function receiveApproval(address _from, uint256 _value, bytes4 _signature);
}

contract TokenERC20 {
    // Public variables of the token
    string public name;
    string public symbol;
    uint256 public decimals = 18;
    // 18 decimals is the strongly suggested default, avoid changing it
    uint256 public totalSupply;

    // This creates an array with all balances
    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint256)) public allowance;

    // This generates a public event on the blockchain that will notify clients
    event Transfer(address indexed _from, address indexed _to, uint256 _value);

    // This notifies clients about the amount burnt
    event Burn(address indexed _from, uint256 _value);

    /**
     * Constructor function
     * Initializes contract with initial supply tokens to the creator of the contract
     */
    function TokenERC20(
        uint256 initialSupply,
        string tokenName,
        string tokenSymbol
    ) public {
        totalSupply = initialSupply * 10 ** uint256(decimals); // Update total supply
        balances[_from] = initialSupply; // Give the creator all
    }
}
```

- ① Under the dropdown showing “TokenERC20”, add a number (total amount of tokens to issue) and two strings (the latter is the token symbol)
- ② Add enough gas (top right, try 30)
- ③ Click Create and check whether MetaMask needs confirmation



# Interacting with the contract

- ① A new interface will pop up on the bottom right corner of the IDE

The screenshot shows a blockchain IDE interface. At the top, there is a header bar with the text "0 pending transactions" and three icons: a square, a document, and a play button. Below this is a search bar containing the text "TokenERC20 at 0xaca5...ee675 (blockchain)". To the right of the search bar is a close button (an "X"). The main area displays a list of function names, each with a parameter list in a separate box. The functions listed are:

- totalSupply
- symbol
- name
- decimals
- allowance address , address
- balanceOf address
- transferFrom address \_from, address \_to,
- burnFrom address \_from, uint256 \_val
- approve address \_spender, uint256 \_
- approveAndCall address \_spender, uint256 \_
- transfer address \_to, uint256 \_value
- burn uint256 \_value

At the bottom of the interface, there is a toolbar with various icons for navigation and operations.



# Table of Contents

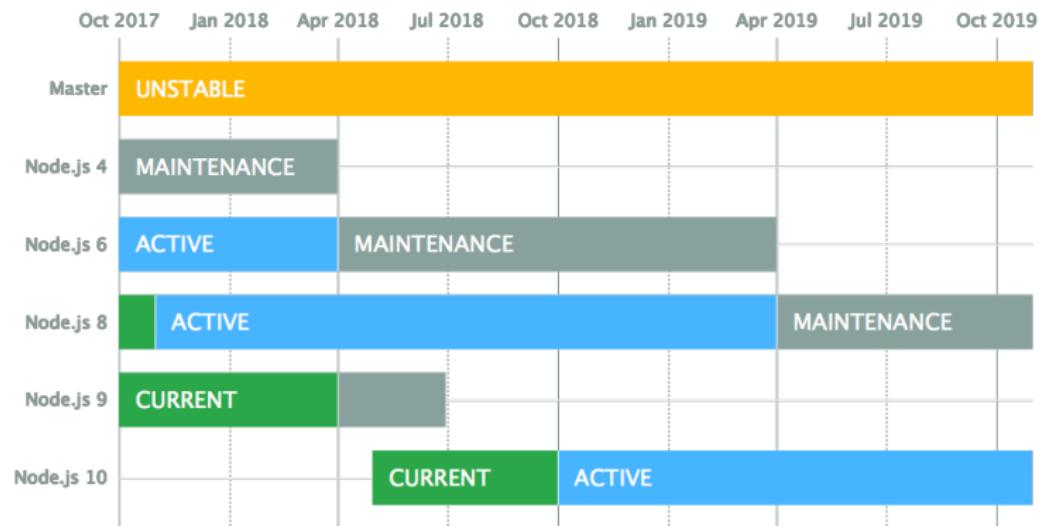
- 1 What is ConsenSys?
- 2 Introduction to Ethereum
- 3 Setting a Development Machine
- 4 ConsenSys Stack Overview
- 5 Hands-on Transactions & Smart Contracts
- 6 Developing with Truffle
- 7 Private blockchain experiment



# Required dependency: node.js > 5

Install version 8 –in LTS maintenance until December 2019.

Run a script from <https://nodejs.org/en/download/package-manager/>



# Installing Truffle

## YOUR ETHEREUM SWISS ARMY KNIFE

Truffle is the most popular development framework for Ethereum with a mission to make your life a whole lot easier.



4,734



594



join chat

INSTALL VIA NPM

```
$ npm install -g truffle
```

Requires NodeJS 5.0+. Works on Linux, macOS, or Windows.

DOCUMENTATION

TUTORIALS

Don't know where to start? Get yourself a [Truffle Box!](#)



# Let's build our first Dapp with Truffle

<http://truffleframework.com/tutorials/pet-shop>



# Let's build an ERC20 Token Contract with Truffle

[http://truffleframework.com/tutorials/  
robust-smart-contracts-with-openzeppelin](http://truffleframework.com/tutorials/robust-smart-contracts-with-openzeppelin)



# Table of Contents

- 1 What is ConsenSys?
- 2 Introduction to Ethereum
- 3 Setting a Development Machine
- 4 ConsenSys Stack Overview
- 5 Hands-on Transactions & Smart Contracts
- 6 Developing with Truffle
- 7 Private blockchain experiment



Let's create our own  
permission-based private  
Blockchain  
based on Ethereum!



# Create a PoA chain with Parity

- PoA: Proof of Authority



# Create a PoA chain with Parity

- PoA: Proof of Authority
- PoA is another type of consensus algorithm (not PoW), with no mining required



# Create a PoA chain with Parity

- PoA: Proof of Authority
- PoA is another type of consensus algorithm (not PoW), with no mining required
- Less computationally intensive, more secure for small networks, faster



# Create a PoA chain with Parity

- PoA: Proof of Authority
- PoA is another type of consensus algorithm (not PoW), with no mining required
- Less computationally intensive, more secure for small networks, faster
- The Kovan test network, Hyperledger and Ripple run on a PoA
- Parity supports two PoA consensus algorithm: Aura, and Tendermint (experimental)



# Create a PoA chain with Parity

- PoA: Proof of Authority
- PoA is another type of consensus algorithm (not PoW), with no mining required
- Less computationally intensive, more secure for small networks, faster
- The Kovan test network, Hyperledger and Ripple run on a PoA
- Parity supports two PoA consensus algorithm: Aura, and Tendermint (experimental)
- Let's follow Parity's Demo PoA tutorial
- Simple Hands-on at  
<https://github.com/marclijour/parity-poa-tutorial>



# Parity's Demo PoA tutorial

## Objectives:

- ① Setup two connected nodes on one machine (for demo)
- ② Gain familiarity with Parity (UI and command line)
- ③ Gain a better understanding of diverse types of blockchain (public/private, permissionless/permission-based) and different types of consensus algorithms



# Parity's Demo PoA tutorial

## Step 1: download the files

This [tutorial](#) assumes than you have installed Parity. Instructions are shown for a machine running Linux Ubuntu. The first step consists in cloning the GitHub repo in your machine. You'll run command from within that directory.

```
$ git clone https://github.com/marclijour/parity-poa-tutorial.git
```



# Parity's Demo PoA tutorial

## Step 2: create nodes and accounts

From the “parity-poa-tutorial” directory, open two terminals and type one line in each:

```
$ parity --config node0.starthere  
$ parity --config node1.starthere
```

Open another console and run these scripts:

```
$ ./create_first_authority_address_on_node0.sh  
$ ./create_second_authority_address_on_node1.sh  
$ ./create_user__address_on_node0.sh
```



# Parity's Demo PoA tutorial

## Step 3: start the chain on PoA

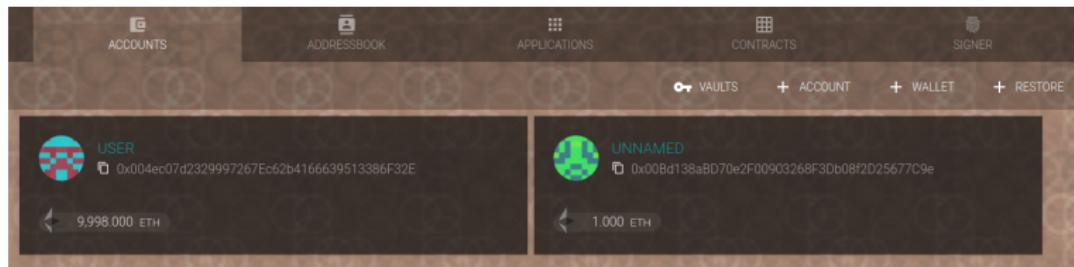
In two separate terminals, restart parity with this new configuration.

```
$ parity --config node0.toml  
$ parity --config node1.toml
```



## Parity's Demo PoA tutorial – Step 4: setup the Parity UI

Open two different windows or tabs in your browser for node 0 (at <http://localhost:8181>) and node 1 (at <http://localhost:8182>).



Restore the accounts as above:

- on node 0: node0 (password = node0), and user (password = user)
- on node 1: node1 (password = node1)



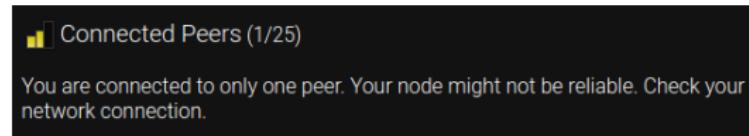
# Parity's Demo PoA tutorial

## Step 4: connect the nodes with each other

Check the console where you started node 0, and look for the Public Node URL. It should resemble something like this: `enode://<long hash>@<IP Address>:<Port Number>`

NETWORK PEERS					ACCEPT NON RESERVED	DROP NON RESERVED	ADD RESERVED	REMOVE RESERVED
ID	Remote Address	Name			Header (ETH)	Difficulty (ETH)	Capabilities	
1	75723705b1b77fe6b7	192.168.0.16:30301	Parity/v1.7.11-stable-af5ed4cf20171228@86.64.1mu-gru/trusts:1.22.1		0x0993...6d1ed1	2.0757224082177246e+40	eth/v6 - eth/v3 - par/v1 - par/v2 - pp/v1	

Go to the Status tab (the leftmost tab) in the Web UI for node 1, and look for the Network Peers section. Click on ADD RESERVED, and copy the URL (including `enode://`).



Check the console output and the Web UI. Both should acknowledge another peer (1/25 Peers instead of 0/25 Peers).



# Parity's Demo PoA tutorial

## Step 5: send transactions

Run the following scripts and watch the balance for each account in the Web UIs.

```
$ send_from_user_to_node0_account.sh  
$ send_from_user_to_node1_account.sh
```

You can also try in a separate console, where you can read the JSON-formatted response.

```
$check_balance_in_node0_account.sh  
$check_balance_in_node1_account.sh
```



# Parity's Demo PoA tutorial

## Step 6: add nodes to the network

Run parity with the right chain specification and let other nodes know (by adding them by enode URL). You just need the demo-spec.json file to get started.

```
$ parity --chain demo-spec.json
```



It's the beginning of a  
rewarding journey...



# Next Steps and Recommended Readings

- Starting on Blockchain: key learning resources
- Fairly exhaustive references from Andreessen Horowitz
- Parity Wiki (e.g. Token Deployment)
- Ethereum White Paper and Wiki
- MOOCs: Udemy (Solidity), edX (Hyperledger)
- Building Blockchain Projects: Building decentralized Blockchain applications with Ethereum and Solidity by Narayan Prusty (2017)  
–check the section on Proof of Authority (PoA)



# Thank you!

Email: [marc.lijour@consensys.net](mailto:marc.lijour@consensys.net)

Twitter: [@marclijour](https://twitter.com/marclijour)



# References

- Afri. (2017). The ethereum-blockchain size will not exceed 1TB anytime soon. Retrieved from <https://dev.to/5chdn/the-ethereum-blockchain-size-will-not-exceed-1tb-anytime-soon-58a>
- Buterin, V. (2017). A prehistory of the ethereum protocol. Retrieved from <https://vitalik.ca/general/2017/09/14/prehistory.html>
- Daniel. (2017). Retrieved from <http://bc.daniel.net.nz>
- Etherscan. (2017). Ethereum chaindata size (geth with fast sync). Retrieved from <https://etherscan.io/chart2/chainedatasizefast>
- Invezz. (2017). Infographic: The story of ethereum. Retrieved from <https://cdn4.benzinga.com/files/images/2017/July/05/invezz-eth-history-base.jpg>
- Reddit. (2017). Ethereum blockchain size...we have a problem. Retrieved from [https://www.reddit.com/r/ethtrader/comments/7axn5g/ethereum\\_blockchain\\_sizewe\\_have\\_a\\_problem/](https://www.reddit.com/r/ethtrader/comments/7axn5g/ethereum_blockchain_sizewe_have_a_problem/)

