

docs / getting-started

Getting started with Geth

Last edited on April 25, 2023

This page explains how to set up Geth and execute some basic tasks using the command line tools. In order to use Geth, the software must first be installed. There are several ways Geth can be installed depending on the operating system and the user's choice of installation method, for example using a package manager, container or building from source. Instructions for installing Geth can be found on the ["Install and Build"](#) pages.

Geth also needs to be connected to a [consensus client](#) in order to function as an Ethereum node. The tutorial on this page assumes Geth and a consensus client have been installed successfully and that a firewall has been configured to block external traffic to the JSON-RPC port `8545` see [Security](#).

This page provides step-by-step instructions covering the fundamentals of using Geth. This includes generating accounts, joining an Ethereum network, syncing the blockchain and sending ether between accounts. This tutorial uses [Clef](#). Clef is an account management tool external to Geth itself that allows users to sign transactions. It is developed and maintained by the Geth team.

Prerequisites

In order to get the most value from the tutorials on this page, the following skills are necessary:

- Experience using the command line
- Basic knowledge about Ethereum and testnets
- Basic knowledge about HTTP and JavaScript
- Basic knowledge of node architecture and consensus clients

Users that need to revisit these fundamentals can find helpful resources relating to the command line [here](#), Ethereum and its testnets [here](#), http [here](#) and Javascript [here](#). Information on node architecture can be found [here](#) and our guide for configuring Geth to connect to a consensus client is [here](#).

Note

If Geth was installed from source on Linux, `make` saves the binaries for Geth and the associated tools in `/build/bin`. To run these programs it is convenient to move them to the top level project directory (e.g. running `mv ./build/bin/* ./`) from `/go-ethereum`. Then `./` must be prepended to the commands in the code snippets in order to execute a particular program,

e.g. `./geth` instead of simply `geth`. If the executables are not moved then either navigate to the `bin` directory to run them (e.g. `cd ./build/bin` and `./geth`) or provide their path (e.g. `./build/bin/geth`). These instructions can be ignored for other installations.

Background

Geth is an Ethereum client written in Go. This means running Geth turns a computer into an Ethereum node. Ethereum is a peer-to-peer network where information is shared directly between nodes rather than being managed by a central server. Every 12 seconds one node is randomly selected to generate a new block containing a list of transactions that nodes receiving the block should execute. This "block proposer" node sends the new block to its peers. On receiving a new block, each node checks that it is valid and adds it to their database. The sequence of discrete blocks is called a "blockchain".

The information provided in each block is used by Geth to update its "state" - the ether balance of each account on Ethereum and the data stored by each smart contract. There are two types of account: externally-owned accounts (EOAs) and contract accounts. Contract accounts execute contract code when they receive transactions. EOAs are accounts that users manage locally in order to sign and submit transactions. Each EOA is a public-private key pair, where the public key is used to derive a unique address for the user and the private key is used to protect the account and securely sign messages. Therefore, in order to use Ethereum, it is first necessary to generate an EOA (hereafter, "account"). This tutorial will guide the user through creating an account, funding it with ether and sending some to another address.

Read more about Ethereum accounts [here](#).

Step 1: Generating accounts

There are several methods for generating accounts in Geth. This tutorial demonstrates how to generate accounts using Clef, as this is considered best practice, largely because it decouples the users' key management from Geth, making it more modular and flexible. It can also be run from secure USB sticks or virtual machines, offering security benefits. For convenience, this tutorial will execute Clef on the same computer that will also run Geth, although more secure options are available (see [here](#)).

An account is a pair of keys (public and private). Clef needs to know where to save these keys so that they can be retrieved later. This information is passed to Clef as an argument. This is achieved using the following command:

```
clef newaccount --keystore geth-tutorial/keystore
```

The specific function from Clef that generates new accounts is `newaccount` and it accepts a parameter, `--keystore`, that tells it where to store the newly generated keys. In this example the

keystore location is a new directory that will be created automatically: `geth-tutorial/keystore`.
Clef will return the following result in the terminal:

```
WARNING!
```

```
Clef is an account management tool. It may, like any software, contain bugs.
```

```
Please take care to
```

- backup your keystore files,
- verify that the keystore(s) can be opened with your password.

```
Clef is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY  
without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR  
PURPOSE. See the GNU General Public License for more details.
```

```
Enter 'ok' to proceed:  
>
```

This is important information. The `geth-tutorial/keystore` directory will soon contain a secret key that can be used to access any funds held in the new account. If it is compromised, the funds can be stolen. If it is lost, there is no way to retrieve the funds. This tutorial will only use dummy funds with no real world value, but when these steps are repeated on Ethereum mainnet is critical that the keystore is kept secure and backed up.

Typing `ok` into the terminal and pressing `enter` causes Clef to prompt for a password. Clef requires a password that is at least 10 characters long, and best practice would be to use a combination of numbers, characters and special characters. Entering a suitable password and pressing `enter` returns the following result to the terminal:

```
-----  
DEBUG[02-10|13:46:46.436] FS scan times                               list="92.081µs" set="1  
INFO [02-10|13:46:46.592] Your new key was generated                       address=0xCe8dBA5e4157  
WARN [02-10|13:46:46.595] Please backup your key file!                     path=keystore:/// .../c  
WARN [02-10|13:46:46.595] Please remember your password!  
Generated account 0xCe8dBA5e4157c2B284d8853afEEea259344C1653
```

It is important to save the account address and the password somewhere secure. They will be used again later in this tutorial. Please note that the account address shown in the code snippets above and later in this tutorials are examples - those generated by followers of this tutorial will be different. The account generated above can be used as the main account throughout the remainder of this tutorial. However in order to demonstrate transactions between accounts it is also necessary to have a second account. A second account can be added to the same keystore by precisely repeating the previous steps, providing the same password.

Step 2: Start Clef

The previous commands used Clef's `newaccount` function to add new key pairs to the keystore. Clef uses the private key(s) saved in the keystore to sign transactions. In order to do this, Clef needs to be started and left running while Geth is running simultaneously, so that the two programs can communicate between one another.

To start Clef, run the Clef executable passing as arguments the keystore file location, config directory location and a chain ID. The config directory was automatically created inside the `geth-tutorial` directory during the previous step. The [chain ID](#) is an integer that defines which Ethereum network to connect to. Ethereum mainnet has chain ID 1. In this tutorial Chain ID 11155111 is used which is that of the Sepolia testnet. It is very important that this chain ID parameter is set to 11155111 - Clef uses the chain ID to sign messages so it must be correct. The following command starts Clef on Sepolia:

```
clef --keystore geth-tutorial/keystore --configdir geth-tutorial/clef --chainid 11155111
```

After running the command above, Clef requests the user to type “ok” to proceed. On typing "ok" and pressing enter, Clef returns the following to the terminal:

```
INFO [02-10|13:55:30.812] Using CLI as UI-channel
INFO [02-10|13:55:30.946] Loaded 4byte database
WARN [02-10|13:55:30.947] Failed to open master, rules disabled
INFO [02-10|13:55:30.947] Starting signer
DEBUG[02-10|13:55:30.948] FS scan times
DEBUG[02-10|13:55:30.970] Ledger support enabled
DEBUG[02-10|13:55:30.973] Trezor support enabled via HID
DEBUG[02-10|13:55:30.976] Trezor support enabled via WebUSB
INFO [02-10|13:55:30.978] Audit logs configured
DEBUG[02-10|13:55:30.981] IPCs registered
INFO [02-10|13:55:30.984] IPC endpoint opened
----- Signer info -----
* intapi_version : 7.0.1
* extapi_version : 6.1.0
* extapi_http : n/a
* extapi_ipc : geth-tutorial/clef/clef.ipc
embeds=146,841 locals=
err="failed stat on ge
chainid=5 keystore=get
list="133.35µs" set="f
file=audit.log
namespaces=account
url=geth-tutorial/clef
```

This result indicates that Clef is running. This terminal should be left running for the duration of this tutorial. If the tutorial is stopped and restarted later Clef must also be restarted by running the previous command.

Step 3: Start Geth

Geth is the Ethereum client that will connect the computer to the Ethereum network. In this tutorial the network is Sepolia, an Ethereum testnet. Testnets are used to test Ethereum client software and smart contracts in an environment where no real-world value is at risk. To start Geth, run the Geth executable file passing argument that define the data directory (where Geth should save blockchain data), signer (points Geth to Clef), the network ID and the sync mode. For this tutorial, snap sync is recommended (see [here](#) for reasons why). The final argument passed to Geth is the `--http` flag.

This enables the http-rpc server that allows external programs to interact with Geth by sending it http requests. By default the http server is only exposed locally using port 8545: `localhost:8545`. It is also necessary to authorize some traffic for the consensus client which is done using `--authrpc` and also to set up a JWT secret token in a known location, using `--jwt-secret`.

The following command should be run in a new terminal, separate to the one running Clef:

```
ial --authrpc.addr localhost --authrpc.port 8551 --authrpc.vhosts localhost --au
```

Running the above command starts Geth. Geth will not sync the blockchain correctly unless there is also a consensus client that can pass Geth a valid head to sync up to. In a separate terminal, start a consensus client. Once the consensus client gets in sync, Geth will start to sync too.

The terminal should rapidly fill with status updates that look similar to those below. To check the meaning of the logs, refer to the [logs page](#).

```
INFO [02-10|13:59:06.649] Starting Geth on sepolia testnet...
INFO [02-10|13:59:06.652] Maximum peer count                      ETH=50 LES=0 total=50
INFO [02-10|13:59:06.655] Using external signer                  url=geth-tutorial/clef
INFO [02-10|13:59:06.660] Set global gas cap                     cap=50,000,000
INFO [02-10|13:59:06.661] Allocated cache and file handles       database=/.../geth-tut
INFO [02-10|13:59:06.855] Persisted trie from memory database     nodes=361 size=51.17Ki
INFO [02-10|13:59:06.855] Initialised chain configuration        config="{ChainID: 1115
INFO [02-10|13:59:06.862] Added trusted checkpoint                block=5,799,935 hash=2
INFO [02-10|13:59:06.863] Loaded most recent local header         number=6,340,934 hash=
INFO [02-10|13:59:06.867] Configured checkpoint oracle            address=0x18CA0E045F0E
INFO [02-10|13:59:06.867] Gasprice oracle is ignoring threshold set threshold=2
WARN [02-10|13:59:06.869] Unclean shutdown detected               booted=2022-02-08T04:2
INFO [02-10|13:59:06.870] Starting peer-to-peer node             instance=Geth/v1.10.15
INFO [02-10|13:59:06.995] New local node record                  seq=1,644,272,735,880
INFO [02-10|13:59:06.996] Started P2P networking                 self=enode://4b80ebd34
INFO [02-10|13:59:06.997] IPC endpoint opened                    url=/.../geth-tutorial
INFO [02-10|13:59:06.998] HTTP server started                    endpoint=127.0.0.1:854
```

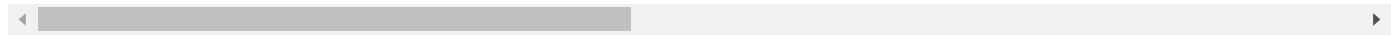
By default, Geth uses snap-sync which download blocks sequentially from a relatively recent block, not the genesis block. It saves the data in files in `/go-ethereum/geth-tutorial/geth/chaindata/`. Once the sequence of headers has been verified, Geth downloads the block bodies and state data before starting the "state healing" phase to update the state for newly arriving data. This is confirmed by the logs printed to the terminal. There should be a rapidly-growing sequence of logs in the terminal with the following syntax:

```
INFO [04-29][15:54:09.238] Looking for peers                      peercount=2 tried=0 static=0
INFO [04-29][15:54:19.393] Imported new block headers             count=2 elapsed=1.127ms number=
INFO [04-29][15:54:19.656] Imported new block receipts            count=698 elapsed=4.464ms numbe
```

This message will be displayed periodically until state healing has finished:

```
INFO [10-20|20:20:09.510] State heal in progress
```

```
accounts=31
```



When state healing is finished, the node is in sync and ready to use.

Sending an empty Curl request to the http server provides a quick way to confirm that this too has been started without any issues. In a third terminal, the following command can be run:

```
curl http://localhost:8545
```

If there is no error message reported to the terminal, everything is OK. Geth must be running and synced in order for a user to interact with the Ethereum network. If the terminal running Geth is closed down then Geth must be restarted again in a new terminal. Geth can be started and stopped easily, but it must be running for any interaction with Ethereum to take place. To shut down Geth, simply press `CTRL+C` in the Geth terminal. To start it again, run the previous command `geth --datadir <other commands>`.

Step 4: Get Testnet Ether

In order to make some transactions, the user must fund their account with ether. On Ethereum mainnet, ether can only be obtained in three ways: 1) by receiving it as a reward for mining/validating; 2) receiving it in a transfer from another Ethereum user or contract; 3) receiving it from an exchange, having paid for it with fiat money. On Ethereum testnets, the ether has no real world value so it 4) can be made freely available via faucets. Faucets allow users to request a transfer of testnet ether to their account.

The address generated by Clef in Step 1 can be pasted into the Paradigm Multifaucet faucet [here](#). The faucet adds Sepolia ETH (not real ETH) to the given address. In the next steps Geth will be used to check that the ether has been sent to the given address and send some of it to the second address created earlier.

Step 5: Interact with Geth

For interacting with the blockchain, Geth provides JSON-RPC APIs. [JSON-RPC](#) is a way to execute specific tasks by sending instructions to Geth in the form of [JSON](#) objects. RPC stands for "Remote Procedure Call" and it refers to the ability to send these JSON-encoded instructions from locations outside of those managed by Geth. It is possible to interact with Geth by sending these JSON encoded instructions directly over Geth's exposed http port using tools like Curl. However, this is somewhat user-unfriendly and error-prone, especially for more complex instructions. For this reason, there are a set of libraries built on top of JSON-RPC that provide a more user-friendly interface for interacting with Geth. One of the most widely used is Web3.js.

Geth provides a Javascript console that exposes the Web3.js API. This means that with Geth running in one terminal, a Javascript environment can be opened in another allowing the user to interact with Geth using Web3.js. There are three transport protocols that can be used to connect the Javascript environment to Geth:

- IPC (Inter-Process Communication): Provides unrestricted access to all APIs, but only works when the console is run on the same host as the geth node.
- HTTP: By default provides access to the `eth`, `web3` and `net` method namespaces.
- Websocket: By default provides access to the `eth`, `web3` and `net` method namespaces.

This tutorial will use the HTTP option. Note that the terminals running Geth and Clef should both still be active. In a new (third) terminal, the following command can be run to start the console and connect it to Geth using the exposed http port:

```
geth attach http://127.0.0.1:8545
```

This command causes the terminal to hang because it is waiting for approval from Clef. Approving the request in the terminal running Clef will lead to the following welcome message being displayed in the Javascript console:

```
Welcome to the Geth JavaScript console!

instance: Geth/v1.10.15-stable/darwin-amd64/go1.17.5
at block: 6354736 (Thu Feb 10 2022 14:01:46 GMT+0100 (WAT))
modules: eth:1.0 net:1.0 rpc:1.0 web3:1.0

To exit, press ctrl-d or type exit
```

The console is now active and connected to Geth. It can now be used to interact with the Ethereum (Sepolia) network.

List of accounts

In this tutorial, the accounts are managed using Clef. This means that requesting information about the accounts requires explicit approval in Clef, which should still be running in its own terminal. Earlier in this tutorial, two accounts were created using Clef. The following command will display the addresses of those two accounts and any others that might have been added to the keystore before or since.

```
eth.accounts;
```

The console will hang, because Clef is waiting for approval. The following message will be displayed in the Clef terminal:

```
----- List Account request-----
A request has been made to list all accounts.
You can select which accounts the caller can see
[x] 0xca57F3b40B42FCce3c37B8D18aDBca5260ca72EC
    URL: keystore:/// .../geth-tutorial/keystore/UTC--2022-02-07T17-19-56.517538000Z--ca57
[x] 0xCe8dBA5e4157c2B284d8853afEEea259344C1653
    URL: keystore:/// .../geth-tutorial/keystore/UTC--2022-02-10T12-46-45.265592000Z--ce8c
-----

Request context:
    NA - ipc - NA

Additional HTTP header data, provided by the external caller:
    User-Agent: ""
    Origin: ""
Approve? [y/N]:
```



Entering `y` approves the request from the console. In the terminal running the Javascript console, the account addresses are now displayed:

```
["0xca57f3b40b42fcce3c37b8d18adbca5260ca72ec", "0xce8dba5e4157c2b284d8853afeeea259344c1653"]
```



It is also possible for this request to time out if the Clef approval took too long - in this case simply repeat the request and approval. Accounts can also be listed directly from Clef by opening a new terminal and running `clef list-accounts --keystore <path-to-keystore>`.

Checking account balance.

Having confirmed that the two addresses created earlier are indeed in the keystore and accessible through the Javascript console, it is possible to retrieve information about how much ether they own. The Sepolia faucet should have sent 0.05 ETH to the address provided, meaning that the balance of one of the accounts should be at least 0.05 ether and the other should be 0. There are other faucets available that may dispense more ETH per request, and multiple requests can be made to accumulate more ETH. The following command displays the account balance in the console:

```
web3.fromWei(eth.getBalance('0xca57F3b40B42FCce3c37B8D18aDBca5260ca72EC'), 'ether')
```



There are actually two instructions sent in the above command. The inner one is the `getBalance` function from the `eth` namespace. This takes the account address as its only argument. By default, this returns the account balance in units of Wei. There are 10^{18} Wei to one ether. To present the result in units of ether, `getBalance` is wrapped in the `fromWei` function from the `web3` namespace. Running this command should provide the following result, assuming the account balance is 1 ETH:

1

Repeating the command for the other (empty) account should yield:

0

Send ether to another account

The command `eth.sendTransaction` can be used to send some ether from one address to another. This command takes three arguments: `from`, `to` and `value`. These define the sender and recipient addresses (as strings) and the amount of Wei to transfer. It is far less error prone to enter the transaction value in units of ether rather than Wei, so the value field can take the return value from the `toWei` function. The following command, run in the Javascript console, sends 0.1 ether from one of the accounts in the Clef keystore to the other. Note that the addresses here are examples - the user must replace the address in the `from` field with the address currently owning 1 ether, and the address in the `to` field with the address currently holding 0 ether.

```
eth.sendTransaction({
  from: '0xca57f3b40b42fcce3c37b8d18adbca5260ca72ec',
  to: '0xce8dba5e4157c2b284d8853afeeea259344c1653',
  value: web3.toWei(0.1, 'ether')
});
```

Note that submitting this transaction requires approval in Clef. In the Clef terminal, Clef will prompt for approval and request the account password. If the password is correctly entered, Geth proceeds with the transaction. The transaction request summary is presented by Clef in the Clef terminal. This is an opportunity for the sender to review the details and ensure they are correct.

```
----- Transaction request-----
to:      0xCe8dBA5e4157c2B284d8853afEEea259344C1653
from:      0xca57F3b40B42FCce3c37B8D18aDBca5260ca72EC [chksum ok]
value:      100000000000000000 wei
gas:      0x5208 (21000)
maxFeePerGas:      2425000057 wei
maxPriorityFeePerGas: 2424999967 wei
nonce:      0x3 (3)
chainid: 0x5
Accesslist

Request context:
    NA - ipc - NA

Additional HTTP header data, provided by the external caller:
    User-Agent: ""
    Origin: ""
-----
Approve? [y/N]:
```

Please enter the password for account 0xca57F3b40B42FCce3c37B8D18aDBca5260ca72EC

After approving the transaction, the following confirmation screen is displayed in the Clef terminal:

```
-----
Transaction signed:
{
  "type": "0x2",
  "nonce": "0x3",
  "gasPrice": null,
  "maxPriorityFeePerGas": "0x908a901f",
  "maxFeePerGas": "0x908a9079",
  "gas": "0x5208",
  "value": "0x2386f26fc10000",
  "input": "0x",
  "v": "0x0",
  "r": "0x66e5d23ad156e04363e68b986d3a09e879f7fe6c84993cef800bc3b7ba8af072",
  "s": "0x647ff82be943ea4738600c831c4a19879f212eb77e32896c05055174045da1bc",
  "to": "0xce8dba5e4157c2b284d8853afeeea259344c1653",
  "chainId": "0xaa36a7",
  "accessList": [],
  "hash": "0x99d489d0bd984915fd370b307c2d39320860950666aac3f261921113ae4f95bb"
}
```

In the Javascript console, the transaction hash is displayed. This will be used in the next section to retrieve the transaction details.

```
"0x99d489d0bd984915fd370b307c2d39320860950666aac3f261921113ae4f95bb"
```

It is also advised to check the account balances using Geth by repeating the instructions from earlier. At this point in the tutorial, the balances of the two accounts in the Clef keystore should have changed by ~0.1 ETH (the sender's balance will have decremented by a little over 0.1 ETH because some small amount was paid in transaction gas).

Checking the transaction hash

The transaction hash is a unique identifier for this specific transaction that can be used later to retrieve the transaction details. For example, the transaction details can be viewed by pasting this hash into the [Sepolia block explorer](#). The same information can also be retrieved directly from the Geth node. The hash returned in the previous step can be provided as an argument to `eth.getTransaction` to return the transaction information:

```
eth.getTransaction('0x99d489d0bd984915fd370b307c2d39320860950666aac3f261921113:
```



This returns the following response (although the actual values for each field will vary because they are specific to each transaction):

```
{
  accessList: [],
  blockHash: "0x1c5d3f8dd997b302935391b57dc3e4fffd1fa2088ef2836d51f844f993eb39c4",
  blockNumber: 6355150,
  chainId: "0xaa36a7",
  from: "0xca57f3b40b42fcce3c37b8d18adbca5260ca72ec",
  gas: 21000,
  gasPrice: 2425000023,
  hash: "0x99d489d0bd984915fd370b307c2d39320860950666aac3f261921113ae4f95bb",
  input: "0x",
  maxFeePerGas: 2425000057,
  maxPriorityFeePerGas: 2424999967,
  nonce: 3,
  r: "0x66e5d23ad156e04363e68b986d3a09e879f7fe6c84993cef800bc3b7ba8af072",
  s: "0x647ff82be943ea4738600c831c4a19879f212eb77e32896c05055174045da1bc",
  to: "0xce8dba5e4157c2b284d8853afeeea259344c1653",
  transactionIndex: 630,
  type: "0x2",
  v: "0x0",
  value: 1000000000000000000
}
```

Using Curl

Up to this point this tutorial has interacted with Geth using the convenience library Web3.js. This library enables the user to send instructions to Geth using a more user-friendly interface compared to sending raw JSON objects. However, it is also possible for the user to send these JSON objects directly to Geth's exposed HTTP port. Curl is a command line tool that sends HTTP requests. This part of the tutorial demonstrates how to check account balances and send a transaction using Curl.

Checking account balance

The command below returns the balance of the given account. This is a HTTP POST request to the local port 8545. The `-H` flag is for header information. It is used here to define the format of the incoming payload, which is JSON. The `--data` flag defines the content of the payload, which is a JSON object. That JSON object contains four fields: `jsonrpc` defines the spec version for the JSON-RPC API, `method` is the specific function being invoked, `params` are the function arguments, and `id` is used for ordering transactions. The two arguments passed to `eth_getBalance` are the account address whose balance to check and the block to query (here `latest` is used to check the balance in the most recently mined block).

```
curl -X POST http://127.0.0.1:8545 \
  -H "Content-Type: application/json" \
  --data '{"jsonrpc":"2.0", "method":"eth_getBalance", "params":["0xca57f3b40b
```

A successful call will return a response like the one below:

```
{"jsonrpc": "2.0", "id": 1, "result": "0xc7d54951f87f7c0"}
```

The balance is in the `result` field in the returned JSON object. However, it is denominated in Wei and presented as a hexadecimal string. There are many options for converting this value to a decimal in units of ether, for example by opening a Python console and running:

```
0xc7d54951f87f7c0 / 1e18
```

This returns the balance in ether:

```
0.8999684999998321
```

Checking the account list

The curl command below returns the list of all accounts.

```
curl -X POST http://127.0.0.1:8545 \  
-H "Content-Type: application/json" \  
--data '{"jsonrpc": "2.0", "method": "eth_accounts", "params": [], "id": 1}'
```

This requires approval in Clef. Once approved, the following information is returned to the terminal:

```
{"jsonrpc": "2.0", "id": 1, "result": ["0xca57f3b40b42fcce3c37b8d18adbca5260ca72ec"]}
```

Sending Transactions

Sending a transaction between accounts can also be achieved using Curl. Notice that the value of the transaction is a hexadecimal string in units of Wei. To transfer 0.1 ether, it is first necessary to convert this to Wei by multiplying by 10^{18} then converting to hex. 0.1 ether is

"0x16345785d8a0000" in hex. As before, update the `to` and `from` fields with the addresses in the Clef keystore.

```
curl -X POST http://127.0.0.1:8545 \  
-H "Content-Type: application/json" \  
--data '{"jsonrpc": "2.0", "method": "eth_sendTransaction", "params": [{"from"
```




This requires approval in Clef. Once the password for the sender account has been provided, Clef will return a summary of the transaction details and the terminal that made the Curl request will display a

response containing the transaction hash.

```
{ "jsonrpc": "2.0", "id": 5, "result": "0xac8b347d70a82805edb85fc136fc2c4e77d31677c2f9e4e7950e6" }
```

Summary

This tutorial has demonstrated how to generate accounts using Clef, fund them with testnet ether and use those accounts to interact with Ethereum (Sepolia) through a Geth node. Checking account balances, sending transactions and retrieving transaction details were explained using the web3.js library via the Geth console and using the JSON-RPC directly using Curl. For more detailed information about Clef, please see [the Clef docs](#).

Downloads		Documentation	
			
© 2013–2023. The go-ethereum Authors Do-not-Track			