# 区块链学堂（第四课）：以太坊Geth基本命令

**[陈琳](http://www.8btc.com/author/17094" \o "由 陈琳 发布) 2017-03-15 11:28 发布在 [竞争币](http://www.8btc.com/digital-currency" \o "查看 竞争币 中的全部文章) [3](http://www.8btc.com/eth-geth" \l "comment) 2874**

## 进入Geth 命令行模式

在上一篇文章中，我们说到，我们可以用下面命令，建立一个新的私有链

geth --datadir "./" --nodiscover console 2>>geth.log

进入命令行模式，其中参数

* –datadir 代表文件夹地址，
* –nodiscover 代表该链条不希望被其他节点发现，
* console >> geth.log 代表将控制台输出到文件geth.log中去

当然从命令行模式退出，也很简单，只要打入exit, 即可退出

## Geth命令行中的Eth.accounts

我们在命令行输入 eth.accounts 可以看到当前该区块链中共有几个账号，以及每个账号的公钥地址。

IMG_256

这里就要说到以太坊的账户体系了，

在以太坊系统中，状态是由被称为“账户”（每个账户由一个20字节的地址）的对象和在两个账户之间转移价值和信息的状态转换构成的。以太坊的账户包含四个部分：

* 随机数，用于确定每笔交易只能被处理一次的计数器
* 账户目前的以太币余额
* 账户的合约代码，如果有的话
* 账户的存储（默认为空）

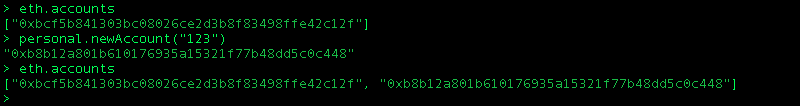
简单地说，每一个以太坊账户都有一对公钥和私钥组成。

* 公钥我们可以理解为就是账户地址，任何其他账户都可以访问该地址
* 私钥可以理解为一段加密过的密码，这一对公钥和私钥共同组成一个唯一标示的以太坊账户。

例如在上节我们建立的第一个以太坊账户 eth.accounts[0] 中，地址0xbcf5b841303bc08026ce2d3b8f83498ffe42c12f 就是公钥，而对密码加密而成的，就是私钥。

## 如何新增账户

我们可以输入命令 personal.newAccount("123") 来新建一个账户，（注意123可以修改为任何别的密码）



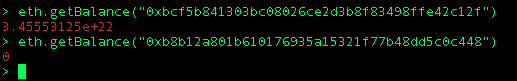
这个时候我们可以看到除了第一个账户0xbcf5b841303bc08026ce2d3b8f83498ffe42c12f之外，还新增了另一个账户 0xb8b12a801b610176935a15321f77b48dd5c0c448, 此时输入eth.accounts, 就可以很轻松的看到有两个账户的公钥地址。

## 如何获取账户的以太币余额

在上一章中我们说过，当以太坊的私链在挖矿时候，所挖到的以太币都会存入第一个以太坊账户中，即eth.accounts[0] 中，而eth.accounts[1]默认是不会有以太币的。这个时候我们可以用下面的命令来查看eth.accounts[0] 中的以太币余额。

eth.getBalance("0xbcf5b841303bc08026ce2d3b8f83498ffe42c12f")

eth.getBalance("0xb8b12a801b610176935a15321f77b48dd5c0c448")

* 其中0xbcf5b841303bc08026ce2d3b8f83498ffe42c12f是第一个账户的地址，
* 0xb8b12a801b610176935a15321f77b48dd5c0c448 是第二个账户的地址 结果如下： 

从上我们可以看得很清楚，挖矿得来的以太币都进了第一个账户，同时每个账户的公钥是该账户的核心。通过公钥我么可以对该账户的以太币进行增删改查各种操作

## 如何在两个账户之间进行以太币转换

前面说过每个账户的公钥（地址）是一切以太坊账户操作的核心，但地址字符串太长，我们用acc0/acc1 分别代表accounts[0]和[1]，另外设置要转移0.01个以太币

> acc0 = eth.accounts[0]

"0xbcf5b841303bc08026ce2d3b8f83498ffe42c12f"

> acc1 = eth.accounts[1]

"0xb8b12a801b610176935a15321f77b48dd5c0c448"

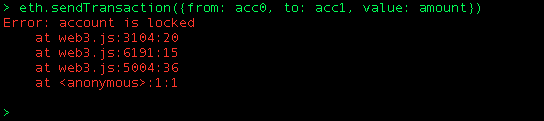
> amount = web3.toWei(0.01)

"10000000000000000"

这个时候我们可以使用eth.sendTransaction来将0.01个以太币从acc0转移到acc1中。

> eth.sendTransaction({from: acc0, to: acc1, value: amount})

结果如下：



这个是以太坊的一个保护机制，每隔一段时间账户就会自动锁定，这个时候任何以太币在账户之间的转换都会被拒绝，除非把该账户解锁.

这个时候我们就需要执行 personal.unlockAccount(acc0) 并输入密码来解锁acc0才可。

> personal.unlockAccount(acc0)

Unlock account 0xbcf5b841303bc08026ce2d3b8f83498ffe42c12f

Passphrase:

true

>

这个时候我们重新执行命令eth.sendTransaction({from: acc0, to: acc1, value: amount}), 结果如下：

> eth.sendTransaction({from: acc0, to: acc1, value: amount})

"0xeea74dd5ff3f1287614d52ebb674edb93e8c5e51e4296835044d3d858d3d9f10"

> eth.getBalance(acc1)

10000000000000000

>

我们可以看到这个时候acc1有了数值10000000000000000, 而不再是之前的0了。但我们明明要给0.01ether币的，为何数值会如此大呢？ 其实是对的，我们只要输入命令web3.fromWei(10000000000000000,"ether") 就可以知道了。

> web3.fromWei(10000000000000000,"ether")

"0.01"

### 为什么呢，这个就涉及到以太坊的基本单位了，我们下章讲解.

———————————————————————————-

## Ether币的基本单位

Ether币最小的单位是Wei，也是命令行默认的单位, 然后每1000个进一个单位，依次是

* kwei (1000 Wei)
* mwei (1000 KWei)
* gwei (1000 mwei)
* szabo (1000 gwei)
* finney (1000 szabo)
* ether (1000 finney)

简单地说就是就是1 以太币 = 1000000000000000000 Wei （这就是上一站章中为何我们转移0.01个以太币，结果却显示很长的原因）

### 如何进行ether 和 Wei之间的转换

Ether–> Wei：web3.toWei

> web3.toWei(1)

"1000000000000000000"

> web3.toWei(1.3423423)

"1342342300000000000"

> web3.toWei(0.00034)

"340000000000000"

>

Wei –> Ether: web3.fromWei

> web3.fromWei(10000000000000000)

"0.01"

> web3.fromWei(1000000000000000000)

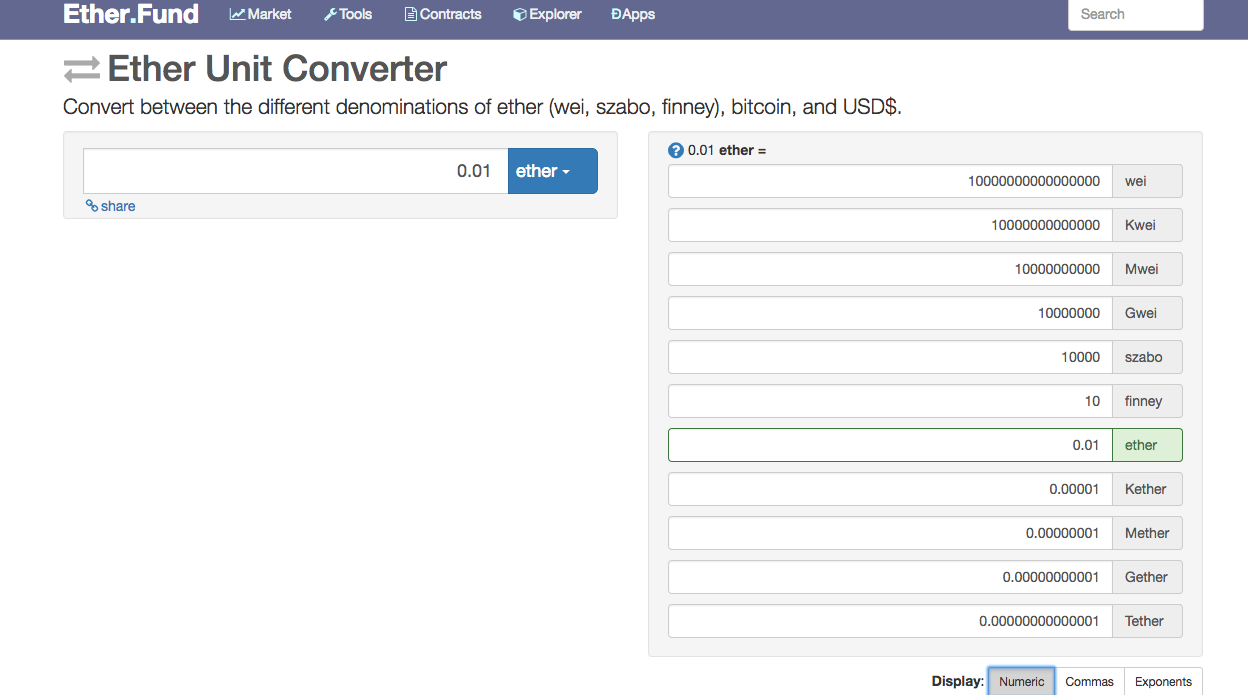
"1"

>

## 一个以太币各单位之间的转换工具

[http://ether.fund/tool/converter](http://ether.fund/tool/converter" \t "http://www.8btc.com/_blank)

使用很简单，输入各种单位，就可以自动得到各种转换结果，例如输入0.01ether 可以得到多少Wei, 多少finney等。



## 开始挖矿 & 停止挖矿

> miner.start() //开始挖矿

true

> miner.stop() //停止挖矿

true

>

## 部署合约

注意合约部署的时候，以太坊的私有链必须处在挖矿进行的状态，否则合约部署将不会生效

* 我们在命令行中，首先unlock(eth.accounts[0]),因为部署合约需要消耗gas，也就是以太币。而之前说过由于保护机制，不解锁账户，是不会允许任何以太币流出的。

> personal.unlockAccount(acc0)

Unlock account 0xbcf5b841303bc08026ce2d3b8f83498ffe42c12f

Passphrase:

true

>

* 然后我们复制黏贴下面代码到geth 命令行中。

var a\_demotypesContract = web3.eth.contract([{"constant":false,"inputs":[{"name":"a","type":"uint256"}],"name":"f","outputs":[{"name":"b","type":"uint256"}],"payable":false,"type":"function"}]);

var a\_demotypes = a\_demotypesContract.new(

{

from: web3.eth.accounts[0],

data: '0x6060604052341561000c57fe5b5b60ab8061001b6000396000f30060606040526000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff168063b3de648b14603a575bfe5b3415604157fe5b60556004808035906020019091905050606b565b6040518082815260200191505060405180910390f35b600060006008830290508091505b509190505600a165627a7a7230582010decdc0b0a43b565814fe904eae2544665457d6353c7d906fc2c43c81c867e40029',

gas: '4700000'

}, function (e, contract){

console.log(e, contract);

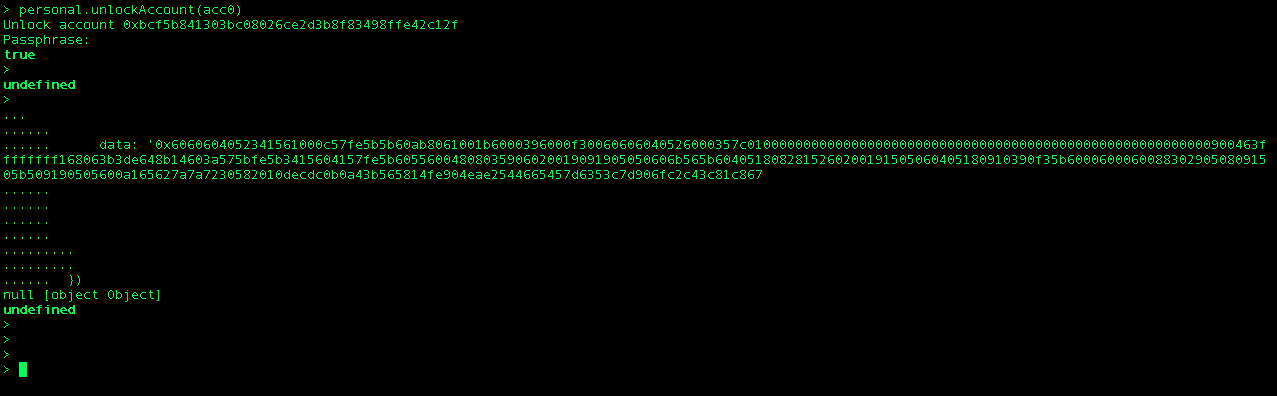
if (typeof contract.address !== 'undefined') {

console.log('Contract mined! address: ' + contract.address + ' transactionHash: ' + contract.transactionHash);

}

})

结果如下图：



等待片刻，会发现合约被部署到挖矿挖出来的区块中了, 按下回车代表成功

[IMG_262](http://7fvhfe.com1.z0.glb.clouddn.com/wp-content/uploads/2017/03/p63.png)

此时输入合约部署的实例a\_demotypes, 可以看到a\_demotypes的详情。

> a\_demotypes

{

abi: [{

constant: false,

inputs: [{...}],

name: "f",

outputs: [{...}],

payable: false,

type: "function"

}],

address: "0x54ed7a5f5a63ddada3bfe83b3e632adabaa5fc2f",

transactionHash: "0x69cde62bcd6458e14f40497f4840f422911d63f5dea2b3a9833e6810db64a1c9",

allEvents: function(),

f: function()

}

>

也可以调用a\_demotypes的方法f, 输入任何数字，会返回8\*n，如输入100，返回800，输入125，返回1000

> a\_demotypes.f.call(100)

800

> a\_demotypes.f.call(125)

1000

##### 由此可见该合约在我们的私有链上部署成功！