プロックチェーン関連文書 Vol.2

Ethereum の動作環境構築について

Vol.2 プライベートネットワークでの動作環境構築

一般社団法人フィンテック研究振興協会

2018 年 2 月 13 日

プライベートネットワークを作成する

◆ Geth を起動する

geth を使用し、ローカル PC 上に Ethereum のプライベートチェーンを立ち上げます。

■ 事前準備

プライベートネットネットワークで geth を起動する為に、次の準備をする必要があります。

- データディレクトリ
- Genesis ファイル

① データディレクトリを作成する

データディレクトリは、送受信したブロックデータやアカウント情報を保存する為に使用します。
任意の場所にディレクトリを作成し、そのディレクトリ内に移動します。
ここでは、ホームディレクトリに「data」を作成します。

```
C:¥Users¥xxx> mkdir data
C:¥Users¥xxx> cd data
```

② Genesis ファイルを作成する

Genesis ファイルは、ブロックチェーンの 1 番最初のブロックの情報を書いた json 形式のテキストファイルです。
①で作成したディレクトリに「genesis.json」を作成します。

```
{
    "config":{
        "chainId":15,
        "homesteadBlock":0,
        "eip155Block":0,
        "eip158Block":0
    },
    "nonce":"0x0000000000000042",
    "timestamp":"0x00",
    "extraData":"0x00",
    "gasLimit":"0xffffffff",
    "difficulty":"0x20000",
    "parentHash":"0x0000000000000000000000000000000000000000000000000000000000000000",
    "mixhash":"0x0000000000000000000000000000000000000000000000000000000000000000",
    "coinbase":"0x0000000000000000000000000000000000000000",
    "alloc":{}
}
```

■ ブロックチェーンを初期化

Genesis.json が作成できたら、ブロックチェーンを初期化します。

```
C:¥Users¥xxx¥data>geth --datadir ./ init ./genesis.json
```

Successfully wrote genesis state メッセージが出力されれば完了です。

```
(省略)
INFO[01-23|17:23:58]Successfully wrote genesis state(省略)
```

■ Geth を起動する

以下のコマンドで geth を起動します。

```
C:¥Users¥xxx¥data> geth --networkid "10" --nodiscover --maxpeers 0 --datadir ./ console 2>> ./geth.log
```

各オプションの内容は以下の通りです。

オプション	説明
--networkid	ネットワーク識別子 0 ～ 3 は予約済みの為、それ以外の数値を指定する
--nodiscover	他ノードとの自動接続を無効にする
--maxpeers	接続できるノード数 0 を指定すると他のノードとは接続しなくなる
--datadir	データディレクトリ
Console	対話型の JavaScript コンソールを起動する
2>>	ログファイル出力パス ※これは geth のオプションではありません

正常に起動すると、次のようなメッセージが表示された後に[>]が表示されます。

```
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.0-unstable/windows-amd64/go1.9.2
modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

>
```

終了する場合は、exit コマンドで終了します。

```
>exit
```

◆ Ether を送金する

Ether をマイニング、送金を行います。

■ アカウントを作成する

新規アカウントを作成します。

Ethereum には以下の 2 種類のアカウントがあります。

- EOA(Externally Owned Account)
- Contract

EOA はユーザーが使用するアカウントで秘密鍵によって管理されるものです。
他のアカウントへ Ether の送金、Contract を実行することができます。
Contract アカウントは、Contract そのものに紐づくアカウントで内部にコントラクト用のプログラムを持っています。EOA から Contract アカウントが呼び出されることでコントラクトが実行されます。

ここでは EOA を作成します。
EOA の作成は personal.newAccount("パスワード")コマンドで行います。
パスワードは作成する EOA のパスワードです。
実行すると、作成された EOA の 20 バイトのアドレスが表示されます。

```
> personal.newAccount("pass0")
"0xf892c48f355608169b25581d8f1846166905f858"
```

eth.acounts でアカウントの確認ができます。

```
> eth.accounts
["0xf892c48f355608169b25581d8f1846166905f858"]
```

送金用にもうひとつアカウントを作成します。

```
> personal.newAccount("pass1")
"0xc81e72f9ab80c7ba5f7a6b86da478c266814c9e9"
> eth.accounts
["0xf892c48f355608169b25581d8f1846166905f858", "0xc81e72f9ab80c7ba5f7a6b86da478c266814c9e9"]
```

アカウント作成は geth コマンドでも行えます。
Passphrase は pass2 とします。

```
C:¥Users¥xxx¥data>geth --datadir ./ account new
Your new account is locked with a password. Please give a password. Do not forget this password.
Passphrase:
Repeat passphrase:
Address: {e4458ccb2e7835f3acce7edc284aebe357897c45}
```

Geth コマンドでアカウントを確認します。

```
C:¥Users¥xxx¥data>geth --datadir ./ account list
Account #0: {f892c48f355608169b25581d8f1846166905f858}
keystore://d:¥Job¥Ethereum¥data¥keystore¥UTC--2018-01-24T04-20-29.662380600Z--
f892c48f355608169b25581d8f1846166905f858
Account #1: {c81e72f9ab80c7ba5f7a6b86da478c266814c9e9}
keystore://d:¥Job¥Ethereum¥data¥keystore¥UTC--2018-01-24T05-21-36.697586600Z--
c81e72f9ab80c7ba5f7a6b86da478c266814c9e9
Account #2: {e4458ccb2e7835f3acce7edc284aebe357897c45}
keystore://d:¥Job¥Ethereum¥data¥keystore¥UTC--2018-01-24T05-26-11.169643100Z--
e4458ccb2e7835f3acce7edc284aebe357897c45
```

■ マイニング

送金する為の Ether を、マイニングして獲得します。
Ethereum では、マイニング成功時に報酬を受け取るアカウントを Etherbase(coinbase)と言います。
Etherbase は eth.coinbase で確認できます。

```
> eth.coinbase
"0xf892c48f355608169b25581d8f1846166905f858"
```

別のアカウントに切り替える場合は、miner.setEtherbase コマンドを実行します。

```
> miner.setEtherbase(eth.accounts[1])
true
> eth.coinbase
"0xc81e72f9ab80c7ba5f7a6b86da478c266814c9e9"
```

eth.getBalance コマンドで現在のアカウントの残高を確認します。
作成直後のアカウントは Ether を所有していない為、どのアカウントでも実行結果は 0 になります。

```
> eth.getBalance(eth.accounts[0])
0
> eth.getBalance(eth.accounts[1])
0
> eth.getBalance(eth.accounts[2])
0
```

Eth.blockNumber でブロックチェーンのブロック数を確認します。
まだマイニングしていないので、0 となります。

```
> eth.blockNumber
0
```

マイニングを開始します。
Miner.start(thread_num)コマンドで開始します。thread_num はマイニングを行うスレッド数を指定します。指定しない場合は、動作環境での CPU コア数に設定されます。
ここでは thread_num を 1 にしてコマンドを実行します。

```
> miner.start(1)
null
```

マイニングされているかの確認は eth.mining コマンドで行えます。
Eth.hashtare コマンドでハッシュレート、eth.blockNumber でブロック高を確認できます。

```
> eth.mining
true
> eth.hashrate
40467
> eth.blockNumber
295
```

Miner.stop コマンドでマイニングを停止します。
マイニングを停止すると、eth.mining は false になります。

```
> miner.stop()
True
> eth.mining
false
```

Etherbase の残高を確認します。このコマンドで表示される数値の単位は wei です。
wei は Ethereum における最小の単位で 1ether = 10^18wei(=1,000,000,000,000,000,000wei)になります。

```
> eth.getBalance(eth.coinbase)
1.515e+21
```

wei だとわかりにくいので、ether に変換します。

```
> web3.fromWei(eth.getBalance(eth.coinbase), "ether")
1515
```

■ Ether の送金

eth.accounts[0]から eth.accounts[1]に 10ether 送金します。
トランザクションの発行は有料になる為、誤って実行できないように通常はロックされています。
使用時にロックを解除する必要があります。Personal.unlockAccount コマンドでロックを解除します。

```
> personal.unlockAccount(eth.accounts[0])
Unlock account 0xf892c48f355608169b25581d8f1846166905f858
Passphrase:
true
```

eth.sendDtansction コマンドで送金を行います。
結果は発行したトランザクションの ID です。

```
> eth.sendTransaction({from:eth.accounts[0], to:eth.accounts[1], value:web3.toWei(10, "ether")})
"0x4fe1761ed01f130bab62631fc15247bbba068f3edeea21e920b4f555992cd989"
```

送金先の eth.accounts[1]の残高を確認すると 0 になっています。

```
> eth.getBalance(eth.accounts[1])
0
```

sendTransaction でトランザクションを発行しただけでは、処理は実行されません。
ブロックチェーンではブロックの中にそのトランザクションが取り込まれる時にトランザクションの内容が実行されます。
eth.getTransaction コマンドで発行されたトランザクションの内容が確認できます。
BlockNumber が null になっているものはブロックに入っていない（未処理）ことを表します。

```
> eth.getTransaction("0x4fe1761ed01f130bab62631fc15247bbba068f3edeea21e920b4f555992cd989")
{
  blockHash: "0x0000000000000000000000000000000000000000000000000000000000000000",
  blockNumber: null,
  from: "0xf892c48f355608169b25581d8f1846166905f858",
  gas: 90000,
  gasPrice: 18000000000,
  hash: "0x4fe1761ed01f130bab62631fc15247bbba068f3edeea21e920b4f555992cd989",
  input: "0x",
  nonce: 0,
  r: "0x1cc353c88436198ff99b2e515dc2db1d158ba9ca4005b5257df6c94e2233eafd",
  s: "0x7681dde1fbe663c1ae8930d3d738a8497fc00899cfdc8876d36c74eb9224870a",
  to: "0xc81e72f9ab80c7ba5f7a6b86da478c266814c9e9",
  transactionIndex: 0,
  v: "0x41",
  value: 10000000000000000000
}
```

Eth.pendingTransactions コマンドでペンディングになっているトランザクションを確認することができます。

```
> eth.pendingTransactions
[{
    blockHash: null,
    blockNumber: null,
    from: "0xf892c48f355608169b25581d8f1846166905f858",
    gas: 90000,
    gasPrice: 18000000000,
    hash: "0x4fe1761ed01f130bab62631fc15247bbba068f3edeea21e920b4f555992cd989",
    input: "0x",
    nonce: 0,
    r: "0x1cc353c88436198ff99b2e515dc2db1d158ba9ca4005b5257df6c94e2233eafd",
    s: "0x7681dde1fbe663c1ae8930d3d738a8497fc00899cfdc8876d36c74eb9224870a",
    to: "0xc81e72f9ab80c7ba5f7a6b86da478c266814c9e9",
    transactionIndex: 0,
    v: "0x41",
    value: 10000000000000000000
}]
```

miner.start コマンドでマイニングを再開して新しくブロックを作成します。

```
> miner.start(1)
Null
```

Eth.pendingTransaction を実行し、トランザクションが表示されなくなったことを確認してから miner.stop コマンドでマイニングを停止します。

```
> eth.pendingTransactions
[]
> miner.stop()
true
> eth.blockNumber
317
```

Eth.getTransaction コマンドでトランザクションを確認すると、blockNumber に値が入ります。

```
> eth.getTransaction("0x4fe1761ed01f130bab62631fc15247bbba068f3edeea21e920b4f555992cd989")
{
   blockHash: "0xcc98b7a938fad68356360bf8aaf87a024502ad0fa5c1495c386777fb12cad168",
   blockNumber: 304,
   from: "0xf892c48f355608169b25581d8f1846166905f858",
   gas: 90000,
   gasPrice: 18000000000,
   hash: "0x4fe1761ed01f130bab62631fc15247bbba068f3edeea21e920b4f555992cd989",
   input: "0x",
   nonce: 0,
   r: "0x1cc353c88436198ff99b2e515dc2db1d158ba9ca4005b5257df6c94e2233eafd",
   s: "0x7681dde1fbe663c1ae8930d3d738a8497fc00899cfdc8876d36c74eb9224870a",
   to: "0xc81e72f9ab80c7ba5f7a6b86da478c266814c9e9",
   transactionIndex: 0,
   v: "0x41",
   value: 10000000000000000000
}
```

送金先の eth.accounts[1]の残高を確認すると 10ether になっています。

```
> eth.getBalance(eth.accounts[1])
10000000000000000000
> web3.fromWei(eth.getBalance(eth.accounts[1]), "ether")
10
```

◆ Ethereum-Wallet を使用する

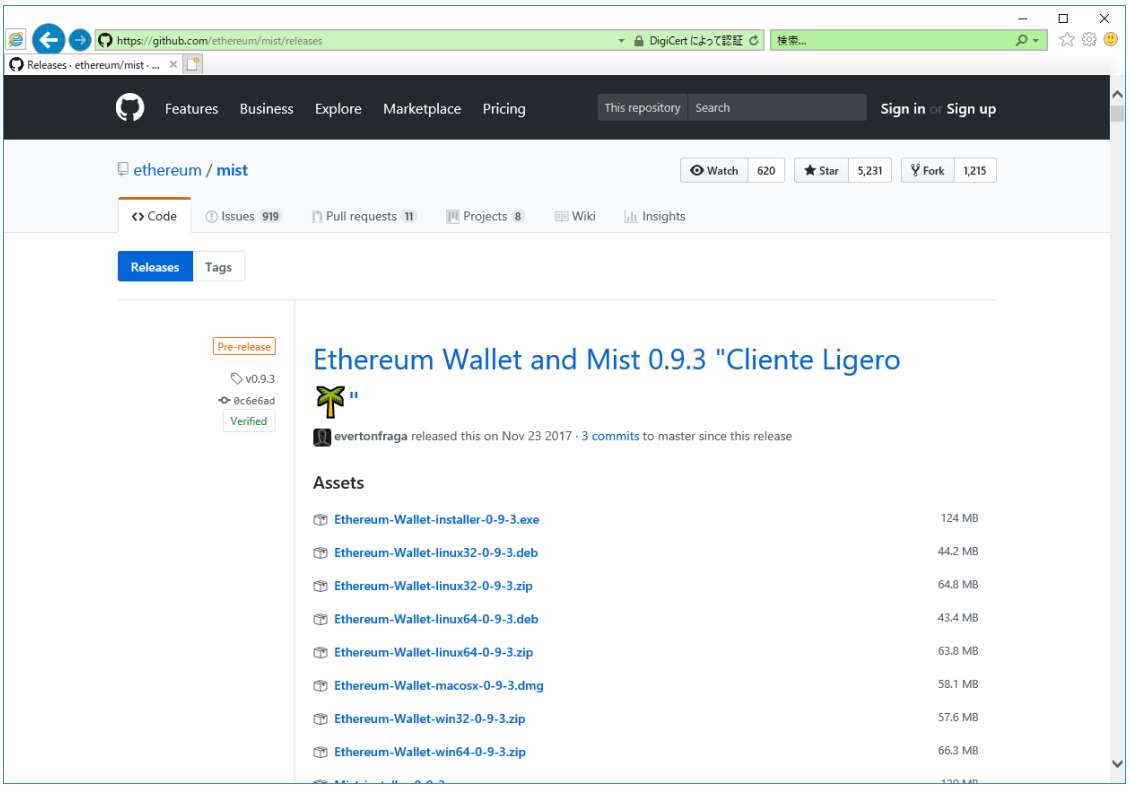
Ethereum-Wallet を使用して、取引を行います。

■ Ethereum-Wallet をダウンロードする

以下のサイトより、Ethereum-Wallet をダウンロードします。

https://github.com/ethereum/mist/releases

以降はバージョン 0.9.3 の Ethereum-Wallet-win64-0-9-3.zip を使用します。



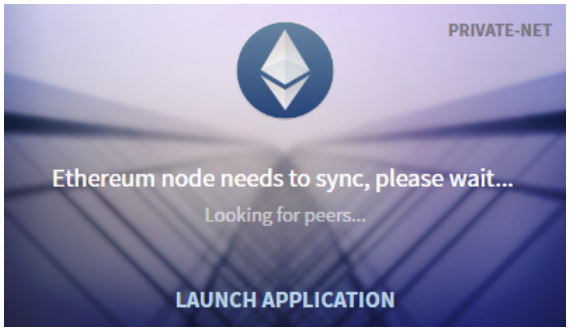
Zip ファイルを任意の場所に展開します。

ブラウザ本体は「Ethereum Wallet.exe」になります。

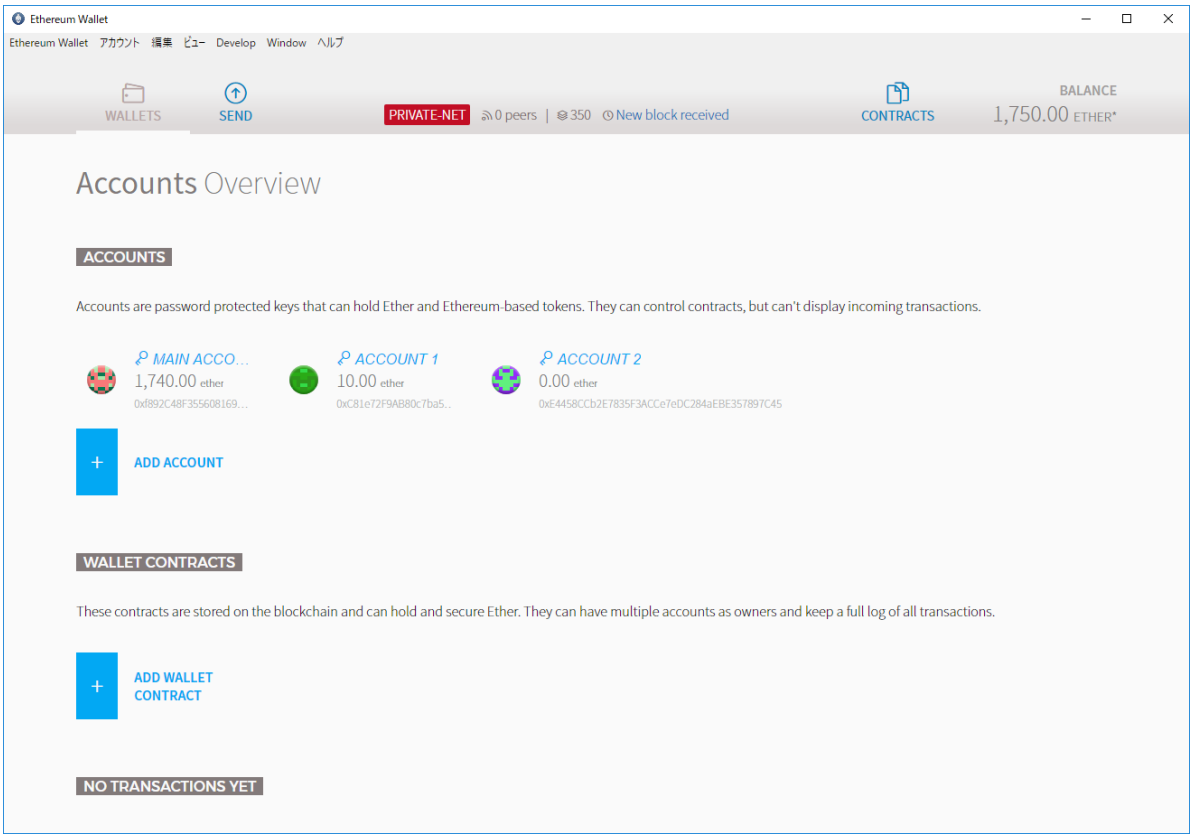
■ Ethereum-Wallet を起動する

「Ethereum Wallet.exe」を起動します。

スプラッシュ画面で LAUNCH APPLICATION が出たらそれをクリックします。



Ethereum-Wallet が立ち上がります。



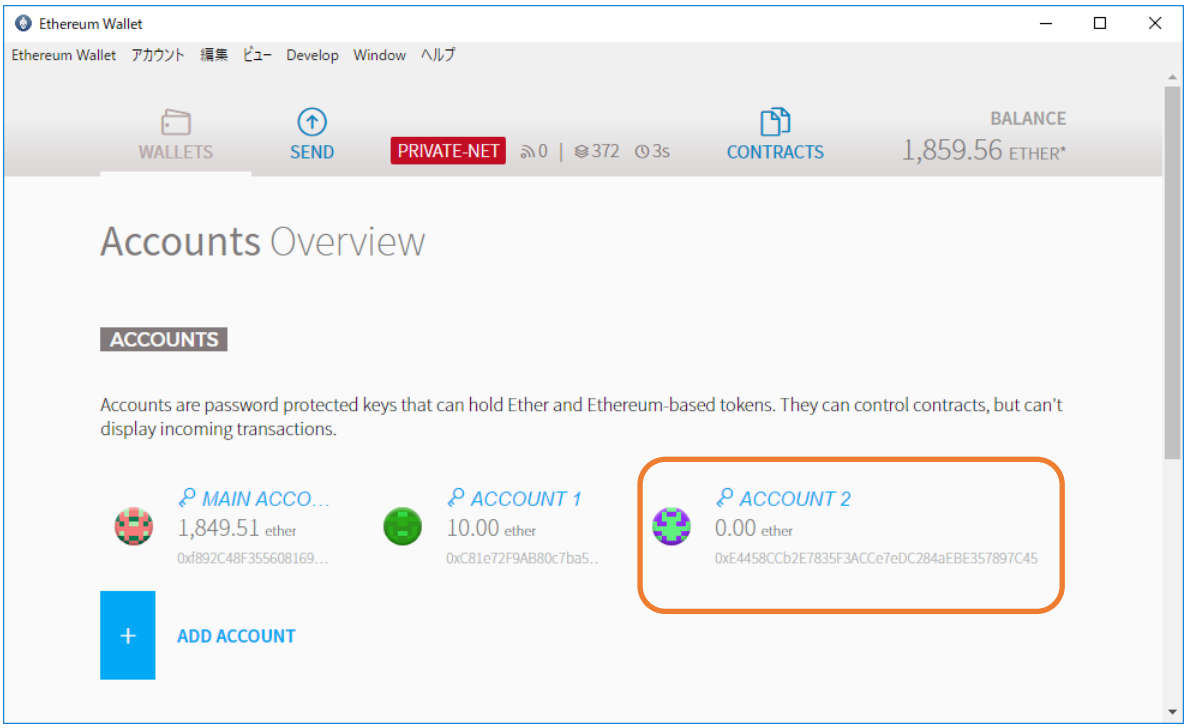
miner.start コマンドでマイニングを開始します。

```
> miner.start(1)
Null
```

■ アカウント間で Ether を送金する

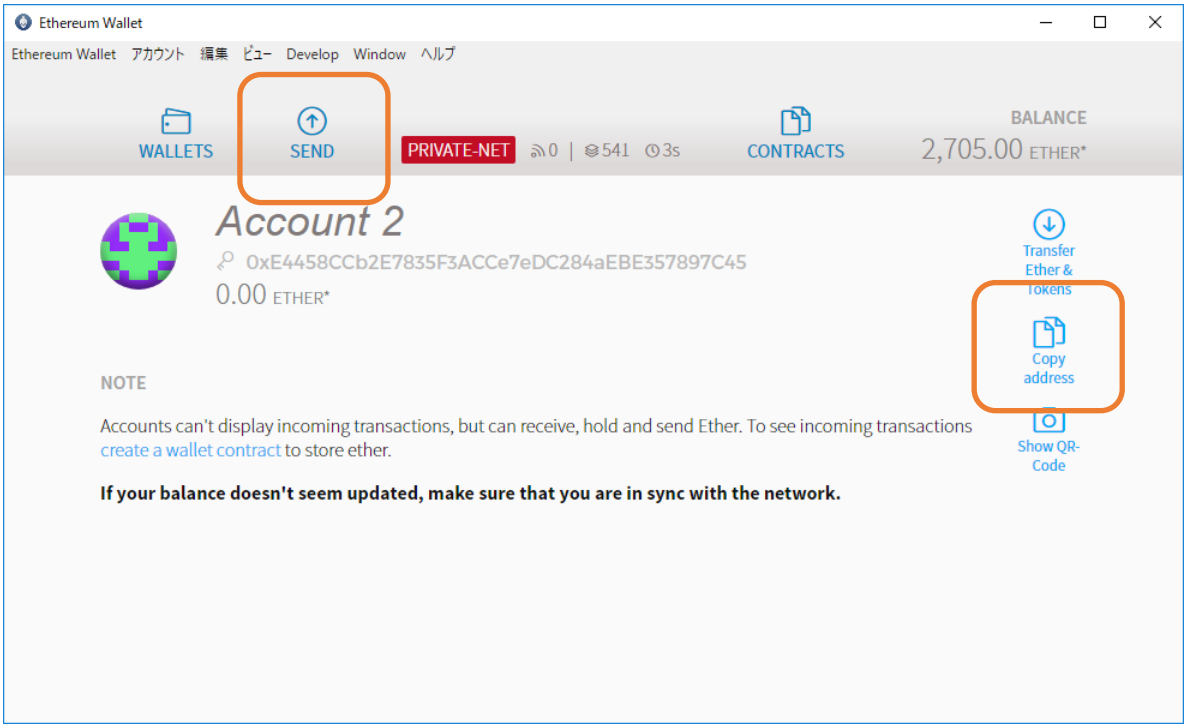
Account1 から Account2 へ送金を行います。

① WALLETS 画面で Account2 をクリックします。

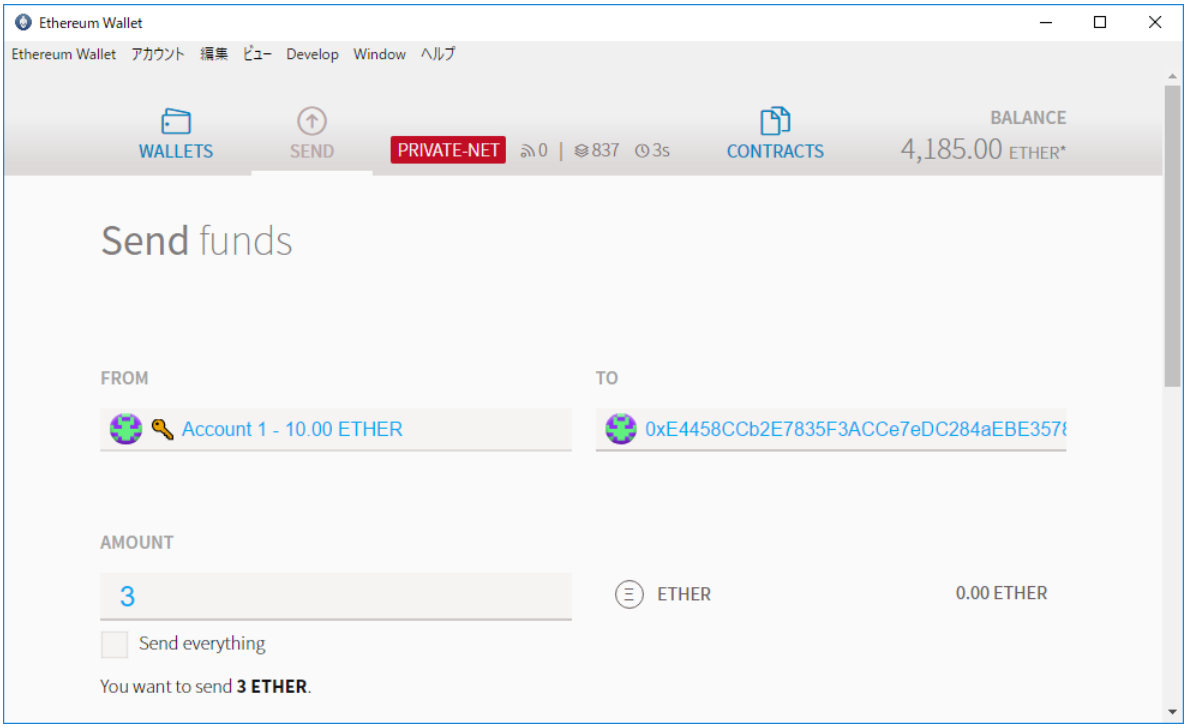


② Account2 の情報が表示されます。

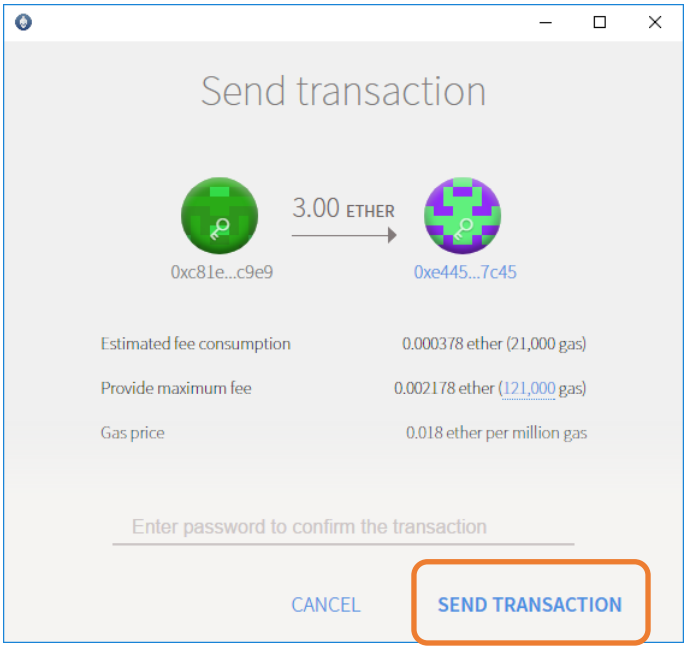
Copy address をクリックして SEND をクリックします。



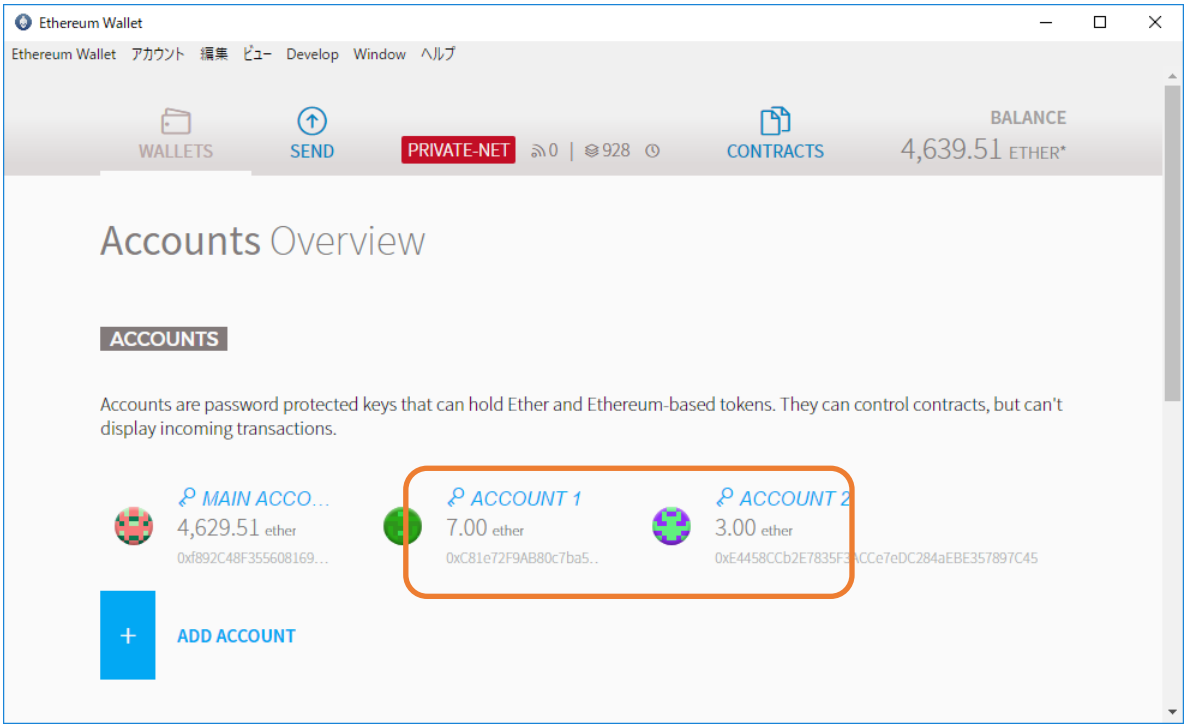
③　Ether の送信元と送信先のアカウントを入力します。
FROM はリストから Account1 を選択、TO は②でコピーしたアドレスを貼り付け（Ctrl+V）します。
AMOUNT に送金金額を入力し、SEND ボタンを押します。



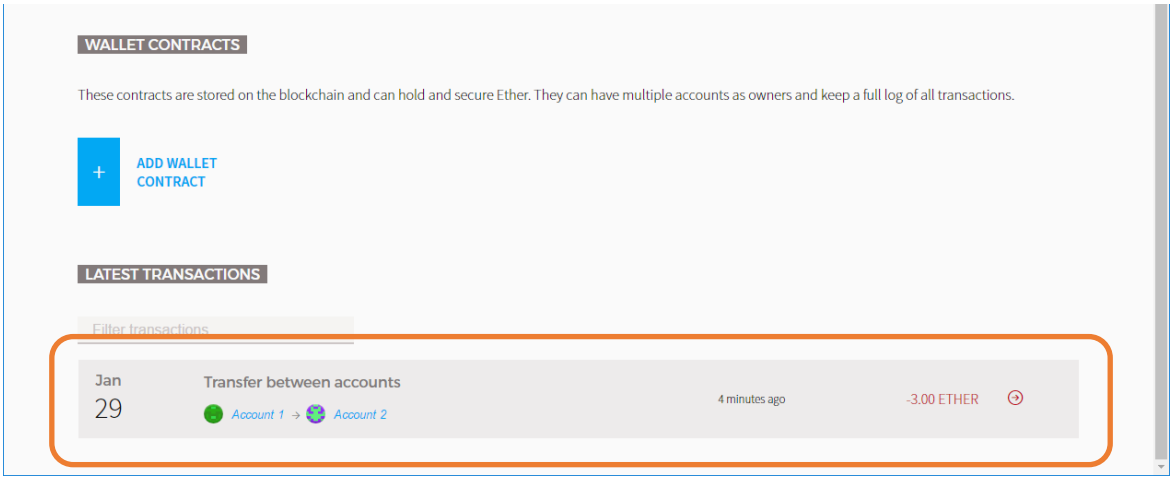
④　トランザクション実行確認画面が出るので、送信元のパスワードを入力後、SEND TRANSACTION ボタンを押します。



⑤　マイニングにより新しいブロックが生成され、Ether が転送されました。



WALLET 画面の最下部にトランザクションの履歴が表示されます。



◆ 複数ノードをつなげてプライベートネットを作成する

端末２台（それぞれ端末 1/端末 2 とする）をつなげてプライベートネットワークを作成します。
各端末の IP アドレスは以下とします。
端末 1：192.168.3.10
端末 2：192.168.3.15

■ 事前準備

端末 1,2 ともに以下の内容にて genesis.json を用意します。

```
{
    "config":{
        "chainId":15,
        "homesteadBlock":0,
        "eip155Block":0,
        "eip158Block":0
    },
    "nonce":"0x0000000000000042",
    "timestamp":"0x00",
    "extraData":"0x00",
    "gasLimit":"0xffffffff",
    "difficulty":"0x20000",
    "parentHash":"0x0000000000000000000000000000000000000000000000000000000000000000",
    "mixhash":"0x0000000000000000000000000000000000000000000000000000000000000000",
    "coinbase":"0x0000000000000000000000000000000000000000",
    "alloc":{}
}
```

Genesis.json が作成できたら、ブロックチェーンを初期化します。

```
C:¥Users¥xxx¥data>geth --datadir ./ init ./genesis.json
```

Successfully wrote genesis state メッセージが出力されれば完了です。

```
(省略)
INFO[01-23|17:23:58]Successfully wrote genesis state(省略)
```

■ Geth を起動する

端末 1,2 ともに以下のコマンドで geth を起動します。

```
C:¥Users¥xxx¥data> geth --networkid "10" --nodiscover --maxpeers 1 --datadir ./ console 2>> ./geth.log
```

※ –maxpeers に接続する最大ピア数を設定します。
ここでは 1 にしてコマンドを実行します。

■ ノードに接続する

端末 1,2 ともに net.peerCount コマンドで現在の接続数を確認します。
まだ接続を行っていない為、どの端末も実行結果は 0 になります。

```
> net.peerCount
0
```

admin.nodeInfo.enode コマンドで端末 1,2 それぞれのノード情報を取得します。

- 端末 1

```
> admin.nodeInfo.enode
"enode://2ed46cf04c0f3350772c6c1d52be2d0fe6763a983c14a42c739a511499cef6fbbc62be7d4b0dbf1fcf52eae17241e9610f745b1cf228bd72964fe78cbd29ffb9@[::]:30303?discport=0"
```

- 端末 2

```
> admin.nodeInfo.enode
"enode://46f28fd2730de83a81284e0d47dd3a3c2911a99bed73d01f685aa5dd7a2cc6994f245d2bd987bcfbf5159e009525c0beea29d539dab89ef8870abbc58d626ae6@[::]:30303?discport=0"
```

admin.addPeer コマンドでノードの接続を行います。
取得したノード情報の"[::]"部分に IP アドレスを記述し、端末 1 に対し端末 2 のノード情報を、端末 2 に対し端末 1 のノード情報を設定します。

- 端末 1

```
> admin.addPeer("enode://46f28fd2730de83a81284e0d47dd3a3c2911a99bed73d01f685aa5dd7a2cc6994f245d2bd987bcfbf5159e009525c0beea29d539dab89ef8870abbc58d626ae6@[192.168.3.15]:30303?discport=0")
true
```

- 端末 2

```
> admin.addPeer("enode://2ed46cf04c0f3350772c6c1d52be2d0fe6763a983c14a42c739a511499cef6fbbc62be7d4b0dbf1fcf52eae17241e9610f745b1cf228bd72964fe78cbd29ffb9@[192.168.3.10]:30303?discport=0")
true
```

それぞれの端末で net.peerCount コマンドで現在の接続数を確認します。
まだ接続を行っていない為、どの端末も実行結果は 1 になります。

```
> net.peerCount
1
```

それぞれの端末で admin.peers コマンドで接続情報を確認します。
端末 1,2 ともに相手側の情報が見え、接続が完了しました。

- 端末 1

```
> admin.peers
[{
      caps: ["eth/63"],
      id: "e169c0a000b66dd647475f50f6e3f01edd494bba9c01982c09938aafd6be9535b3ee5851abfb045ff
af34ec21ecb3720ce7206c4ba0a3b7e2ed3d4c46b1437de",
      name: "Geth/v1.7.3-stable-4bb3c89d/linux-amd64/go1.9",
      network: {
        localAddress: "192.168.3.10:30303",
        remoteAddress: "192.168.3.15:48262"
      },
      protocols: {
        eth: {
          difficulty: 131072,
          head: "0xd966175ff637240bd6965178786497e7365b9935eaa21f2ba0b514d6f22dea8b",
          version: 63
        }
      }
}]
```

- 端末 2

```
> admin.peers
[{
      caps: ["eth/63"],
      id: "2da969556b6c8a749f666ae2c7b0bdc27a5d084107033c9e070734769c7d6f22e0aee176c414005
2b928fa1fbd42a408d0a8ab0d9ae6d196a2a19906a5b17847",
      name: "Geth/v1.8.0-unstable/windows-amd64/go1.9.2",
      network: {
        localAddress: "192.168.3.15:48262",
        remoteAddress: "192.168.3.10:30303"
      },
      protocols: {
        eth: {
          difficulty: 2103872,
          head: "0x63020ec60e5b5e6f3123b440f6343493e442b2bf87b91b6be77e543f8ab5acf7",
          version: 63
        }
      }
}]
```

■ geth 起動時にノードに接続する

ノードが常に固定で決まっている場合は、「static-nodes.json」ファイルを作成することにより、geth 起動時に自動的に接続することができます。

端末 1,2 ともにデータディレクトリに「Static-nodes.json」を作成します。
設定内容は、「admin.addPeer」コマンドで指定したものと同じです。

- 端末 1

```
[
"enode://46f28fd2730de83a81284e0d47dd3a3c2911a99bed73d01f685aa5dd7a2cc699
4f245d2bd987bcfbf5159e009525c0beea29d539dab89ef8870abbc58d626ae6@[192.168.3.15]:30303?disc
port=0"
]
```

- 端末 2

```
[
"enode://2ed46cf04c0f3350772c6c1d52be2d0fe6763a983c14a42c739a511499cef6fbb
c62be7d4b0dbf1fcf52eae17241e9610f745b1cf228bd72964fe78cbd29ffb9@[192.168.3.10]:30303?discpor
t=0"
]
```

※3 台の端末でノードを接続する場合は以下のように設定します。

```
[
"enode://46f28fd2730de83a81284e0d47dd3a3c2911a99bed73d01f685aa5dd7a2cc699
4f245d2bd987bcfbf5159e009525c0beea29d539dab89ef8870abbc58d626ae6@[192.168.3.15]:30303?disc
port=0",
"enode://2ed46cf04c0f3350772c6c1d52be2d0fe6763a983c14a42c739a511499cef6fbb
c62be7d4b0dbf1fcf52eae17241e9610f745b1cf228bd72964fe78cbd29ffb9@[192.168.3.10]:30303?discpor
t=0"
]
```