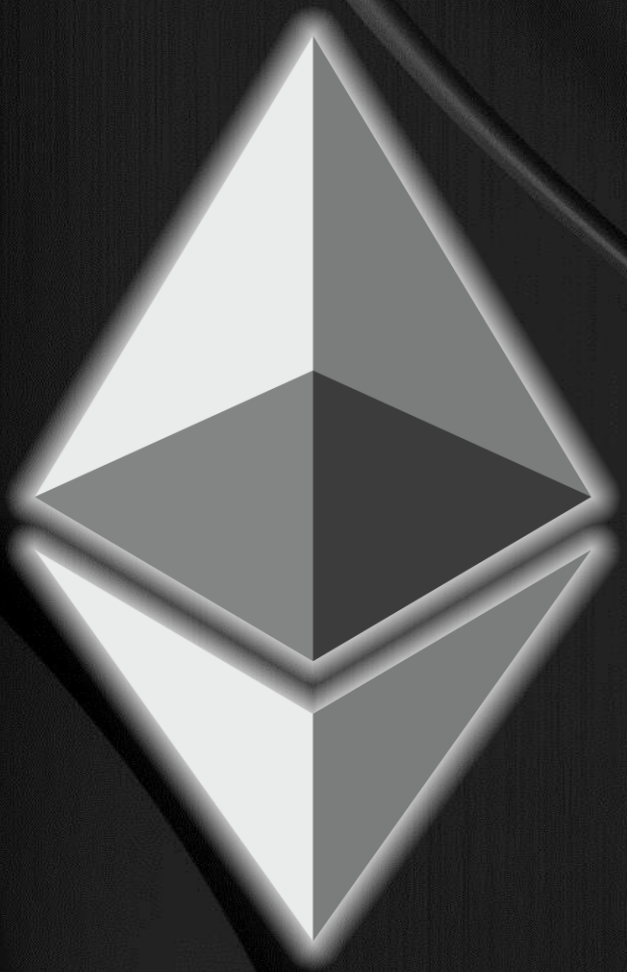


ETHEREUM

Rapid Learning & Just In Time Support

CONTENT

1 INTRODUCTION 5

1.1 Terms 5

2 CLIENTS 8

2.1 GETH 9

2.1.1 Setup 11

2.1.1.1 Install 12

2.1.1.2 Edit PATH Environment Variable 14

2.1.2 Private Network 15

2.1.2.1 Setup 16

2.1.2.2 Connect 18

2.1.3 Account 19

2.1.3.1 Create 19

2.1.3.2 Unlock 20

2.1.3.3 Change Password 21

2.1.3.4 Show All 22

2.1.3.5 Delete 23

2.1.3.6 Set Base Account 24

2.1.3.7 Get balance 25

2.1.4 Ether 26

2.1.4.1 Mine 26

2.1.4.2 Transfer 29

2.1.5 SOLC 31

2.1.5.1 Check if GETH is linked to SOLC 31

2.1.5.2 Link GETH with SOLC 32

2.1.6 Contracts 33

2.1.6.1 Create using online SOLC - Browser Solidity 34

2.1.6.2 Create using online SOLC - Ether Chain 37

2.1.6.3 Create using installed SOLC 40

2.1.6.4 Run by others 46

2.1.6.5 Kill 48

2.2 ETH 49

2.2.1 Setup 49

2.2.1.1 Install 49

2.2.1.2 Edit PATH Environment Variable 51

3 SOLIDITY 52

3.1 Comments 53

3.1.1 Single Line 54

3.1.2 Multi Line 55

3.2 Operators 56

3.2.1 Assignment 56

3.2.2 Comparison 57

3.2.3 Arithmetic 58

3.2.4 Bitwise 59

3.2.5 Logical 60

3.3 Statements 61

3.3.1 Conditional Branching 62

3.3.1.1 if 63
3.3.1.2 else 64
3.3.1.3 else if 65
3.3.2 Conditional Looping 66
3.3.2.1 for 67
3.3.2.2 while 68
3.3.3 Unconditional Jumping 69
3.3.3.1 break 70
3.3.3.2 continue 71
3.4 Functions 72
3.4.1 Name 73
3.4.2 Return values 74
3.4.2.1 None 75
3.4.2.2 Single 76
3.4.2.3 Multiple 77
3.4.3 Parameters 78
3.4.3.1 None 78
3.4.3.2 Multiple 79
3.4.4 Type 80
3.4.4.1 public 81
3.4.4.2 private 82
3.4.4.3 constant 83
3.4.5 Reference 84
3.4.5.1 Internal Function Call 84
3.5 Contracts 85
3.5.1 Name 86
3.5.2 Fields 87
3.5.2.1 public 88
3.5.2.2 private 89
3.5.3 Methods 90
3.5.4 Constructor 91
3.5.5 Inheritance 92
3.5.1 Data Types 94
3.5.2 Scalar 95
3.5.2.1 bool 95
3.5.2.2 int 96
3.5.2.3 uint 97
3.5.2.4 address 98
3.5.2.5 byte 99
3.5.3 Compound 100
3.5.3.1 bytes 101
3.5.3.2 string 102
3.5.3.3 struct 103
3.5.3.4 array 104
4 RELATED TECHNOLOGIES 105
4.1 Chocolatey 106
4.1.1 Install 106
4.1.2 Uninstall 110
4.2 Windows 111

4.2.1 Show Environment Variables 111
4.2.2 Start Command Prompt 112
4.3 Python 113
4.3.1 Install 114
4.4 JSON Formatter 115
5 ERRORS 117
5.1 SyntaxError json.dumps 117
5.2 TypeError: 'greet' is not a function 118

1 Introduction

Info

- Ethereum is decentralized network of nodes which prevents downtime, censorship, fraud or third party interference.
- Node is any computer that runs an Ethereum client.
- Ethereum clients execute applications which are called contracts and can be implemented in JavaScript, C++ or Python.
- Ethereum is based on Bitcoin blockchain technology and uses Ether as its digital currency.
- Using contracts consumes Ether which can be mined with Ethereum clients.

1.1 Terms

Info

- This tutorial explains Ethereum terms.

Releases

- Frontier is the first release of the Ethereum project, tailored specifically for developers.

Clients

- Ethereum clients are applications that are used to connect to Ethereum network and there are three different versions
  - GETH is implemented in Go and it will be used in Mist Browser
  - ETH is implemented in C++ and it will be used for contract development toolset Mix IDE
  - Pyethapp is implemented in Python
- You can use these clients to connect either to
  - private network which should be used for testing since it allows you to easily mine your own ether
  - public network which should be used for production since mining ether, needed to use the network, is much harder

Console

- Connecting to network using `console` keyword allows you to execute JavaScript commands either directly or from a file.
- You can then use these commands to create accounts, start mining, transfer ether between accounts and so on.
- Some of these commands, like creating accounts, can also be executed using clients with additional parameters.

Mining [R]

- Mining is process of
  - creating new Ether which is analogy for how gold is created by mining.
  - securing the network by creating, verifying, publishing and propagating blocks in the blockchain.
- When first command to mine is given your computer first needs to go through a process called "building a DAG".
  It is approximately 1GB big file which takes about 10 minutes create and each Private Network needs its own DAG file.
  As soon as it is created, Geth will start mining automatically adding ether to base account every time a block is mined.
- Mining on private network allows you to easily get Ether to test contracts, each network requiring is own DAG file.
  Mining on the public network requires dedicated computers otherwise getting Ether can take a long time.

Đapps [R]

- Ethereum is a platform that allows people to easily write decentralized applications (Đapps) using blockchain technology.
- Early examples of Đapps include BitTorrent for file sharing and Bitcoin for digital currency.
- Ethereum takes the primary developments used by BitTorrent and Bitcoin, the peer to peer network and the blockchain, and generalizes them in order to allow developers to use these technologies for any purpose.

Ethereum Virtual Machine (EVM) [R]

- The part of the protocol that handles internal state and computation is referred to as the EVM.
- EVM can be thought of as a large decentralized computer containing millions of objects, called "accounts", which have the ability to maintain an internal database, execute code and talk to each other.

Blockchain [R]

- Ethereum blockchain can be described as a
  - blockchain with a built-in programming language
  - consensus-based globally executed virtual machine

Accounts [R]

- There are two types of accounts:
  - Normal or Externally owned account (EOA) is an account controlled by a private key.
    If you own the private key you have the ability to send ether and messages from it.
  - Contract is an account that has its own code, and is controlled by code.
- Both types of accounts have an ether balance.

Transactions [R]

- Contracts interact with each other through an activity called either "calling", "transactions" or "sending messages".
  When contract receives message it can return some data, which can be used by the sender, exactly like calling a function.
- Contracts only fire transactions in response to other transactions that they have received.
  Therefore, all action on the Ethereum is set in motion by user triggering an action by sending a transaction from an EOA.
- The simplest transactions are ether transfer transactions between two EOAs.
  If destination of the transaction is another EOA, then transaction may transfer some ether but otherwise does nothing.
  However, if the destination is a contract, then the contract in turn activates, and automatically runs its code.

Messages

- Message is object which contracts send to each other and contain
  - some quantity of ether (with the primary purpose of paying transaction fees)
  - a byte-array of data of any size
  - the addresses of a sender
  - the addresses of a recipient.

Coins

- Coin is a tradable token that can be
  - used to control access (an entrance ticket)
  - used to represent debt owed by an organization (a bond)
  - used as an exchange of value within a community (a currency)
  - placeholder for an asset held by a third party (a certificate of ownership)
- Ethereum token are censorship proof by having following advantages
  - it's a decentralized service and tokens can be still exchanged even if the original service goes down for any reason
  - code can guarantee that no tokens will ever be created other than the ones set in the original code
  - by having each user hold their own token, this eliminates the scenarios where one single server break-in can result in the loss of funds from thousands of clients.

Gas [R]

- Gas is unit that defines cost of computation. It is a cost of executing operations on Ethereum network.
  In other words for each operation you have to pay fixed amount of Gas to execute it.
- Gas is automatically bought using Ether from the Account that executed the transaction.
  In other words when Account executes a contract it will loose certain amount of Ether which will be used to buy Gas needed to run contract operations.
- The price of Gas in Ether is floating. Amount of Ether needed to buy specific quantity of gas may vary.
- Purpose of Gas is to
  - limit infinite loops
  - prevent DDOS attacks
  - make users pay for network usage
- Additionally for each transaction you define mining fee which is reward for miners.
- This motivates miners to first take care of transactions with highest mining fees.

Ether Denominations

- Ether can also be expressed in denominations listed in the following table.

Ether denominations

1 **ether** =	
1000000000000000000	wei
1000000000000000	Kwei
1000000000000	Mwei
1000000000	Gwei
1000000	szabo
1000	finney
1	ether
0.001	Kether
0.000001	Mether
0.000000001	Gether
0.000000000001	Tether

Kraken [R]

- Kraken is exchange for trading Ether which got its name after mythical sea monster that appeared off the coast of Norway
- You can use Kraken to buy Ether with Bitcoins, Euros or Dollars.

2 Clients

Info

- Ethereum clients are applications that are used to connect to Ethereum network and there are three different versions
  - GETH is implemented in Go and it will be used in Mist Browser
  - ETH is implemented in C++ and it will be used for contract development toolset Mix IDE
  - Pyethapp is implemented in Python
- You can use these clients to connect either to
  - private network which should be used for testing since it allows you to easily mine your own ether
  - public network which should be used for production since mining ether, needed to use the network, is much harder

2.1 GETH

Info **[R]**

- GETH is command line interface for running a full Ethereum node implemented in Go and allows you to
  - mine real ether
  - transfer funds between addresses
  - create contracts and send transactions
  - explore block history
- You can interact with GETH using following interfaces
  - Command line options
  - JavaScript Console available when GETH is launched with console option exposing web3 (Đapp) and admin APIs

Basic GETH Commands

EXAMPLE	DESCRIPTION
geth	Connect to public network.
geth --networkid 123	Connect to new private network.
geth --datadir G:\Temp\Ethereum\datadir	Connect to already created private network.
geth console	Open console after connecting to network.
geth attach	Open additional console which will not contain log lines.
geth 2>> G:\Temp\Ethereum\my.log	Write log into file instead of displaying it.
geth account new	Create new account by providing password.
geth account update 2f5a853427848d5a	Change password of the Account with specified address.
geth account list	Show list of Accounts. For each account index and address is displayed.
geth --etherbase 1	Set base account to account at index 1.
geth --etherbase '0xa4d8e9cae4d04b09'	Set base account to account with address '0xa4d8e9cae4d04b09'.
geth --mine	Start mining ether. Ether is added to base account.
geth --solc "I:/Ethereum/Release/solc.exe"	Specify location of Solidity Compiler.

Console Commands

EXAMPLE	DESCRIPTION
personal.newAccount("mypassword");	Create account with specified password.
eth.accounts;	Show accounts.
myaccount = eth.accounts[0];	Save reference to account into variable.
eth.getBalance(myaccount);	Returns ether balance of 10 as 10000000000000000000.
eth.getBalance(myaccount).toNumber();	Returns ether balance of 10 as 10000000000000000000.
web3.fromWei(eth.getBalance(myaccount), "ether");	Returns ether balance of 10 as 10.
miner.start(8);	Start mining.
admin.sleepBlocks(10);	Wait for 10 blocks to be mined.
miner.stop();	Stop mining.
txpool.status	shows transaction pool
eth.getBlockTransactionCount("pending");	number of pending txs
eth.getBlock("pending", true).transactions	print all pending txs
eth.getCompilers();	Show list of available compilers.
loadScript("/Users/username/gethload.js")	Load JavaScript file.
admin.setSolc("I:/Ethereum/Release/solc.exe");	Link to Solidity Compiler.
myContractInstance.transactionHash	Hash of the transaction, which created the contract.
myContractInstance.address	Defined after contract transaction is mined.

Print all balances

```
function checkAllBalances() {
  var i =0;
  eth.accounts.forEach( function(e){
    console.log("eth.accounts["+i+"]: "+e+" \tbalance: "+web3.fromWei(eth.getBalance(e), "ether")+"ether");
    i++;
  })
};
```

Send Ether

```
> eth.sendTransaction({from:eth.coinbase, to:eth.accounts[1], value: web3.toWei(0.05, "ether")})
> eth.sendTransaction({from: '0x036a03fc47084741f83938296a1c8ef67f6e34fa', to:
'0xa8ade7feab1ece71446bed25fa0cf6745c19c3d5', value: web3.toWei(1, "ether")})
```

- Also, be advised that the amount debited from the source account will be slightly larger than that credited to the target account, which is what has been specified. The difference is a small transaction fee, discussed in more detail later.

2.1.1 Setup

Info

- Following tutorials show how to install and setup GETH Client.

2.1.1.1 Install

Info

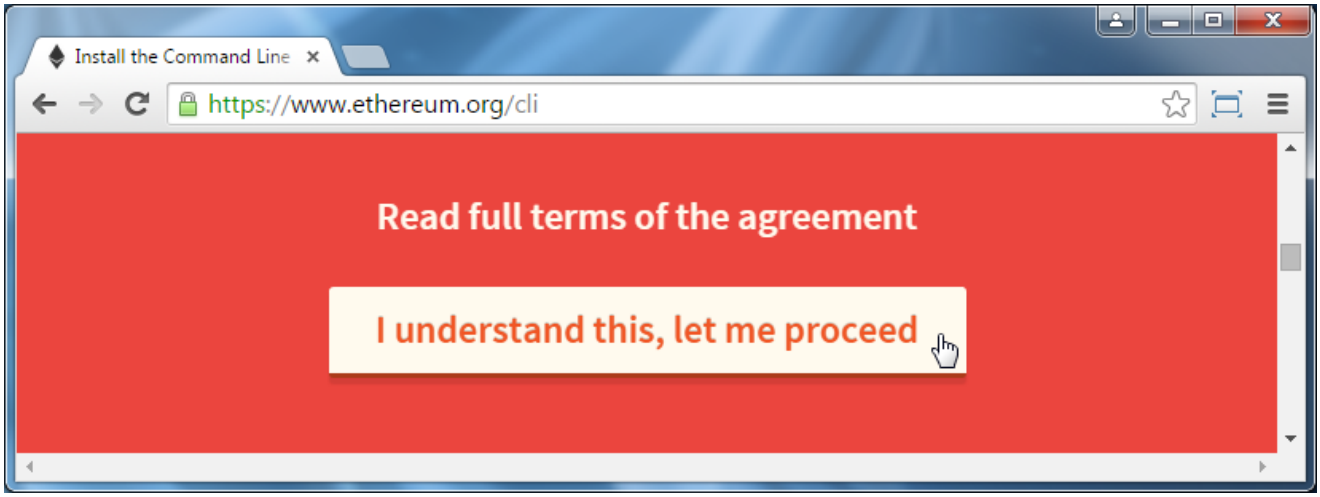
- This tutorial shows how to use Chocolatey to install Ethereum GETH Client.
- GETH Client is Command Line Tools used for interaction with Ethereum.

Read License Agreement

- https://www.ethereum.org/#install-geth
- Install the Command line tools
- I understand this, let me proceed

Install the Command line tools

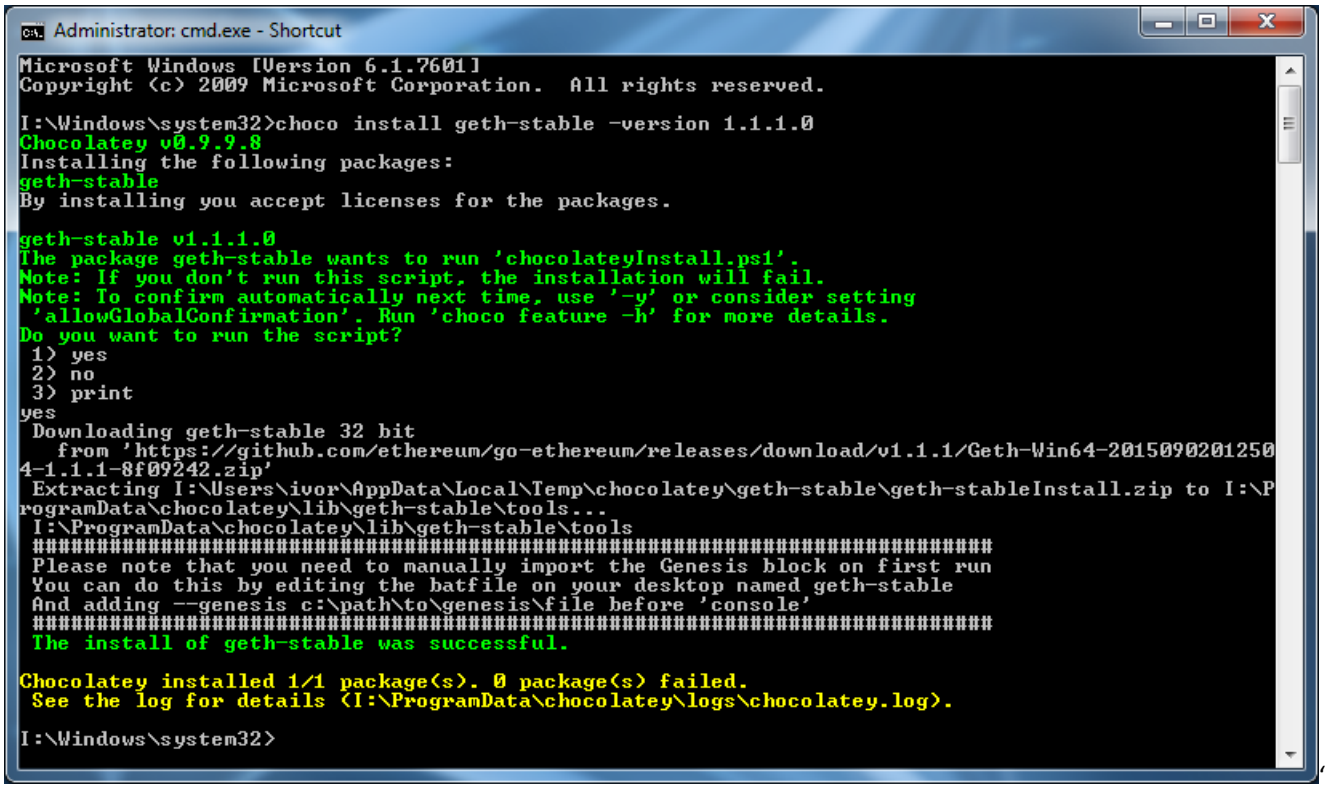
I understand this, let me proceed



Install GETH

- Install Chocolatey
- Start Command Prompt as Administrator
- `choco install geth-stable -version 1.1.1.0`
- `yes`

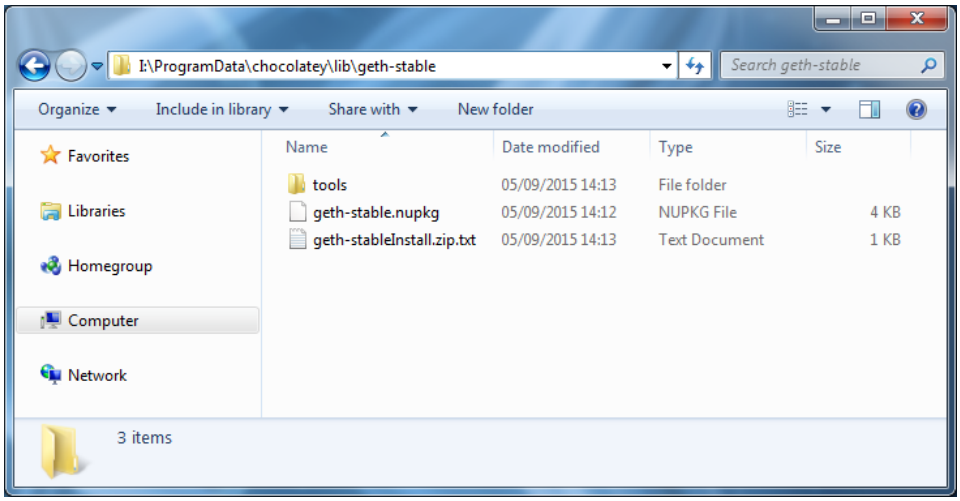
choco install geth-stable -version 1.1.1.0



Results

- GETH will be installed inside geth-stable directory under lib directory where Chocolatey was installed.

Created directory



2.1.1.2 Edit PATH Environment Variable

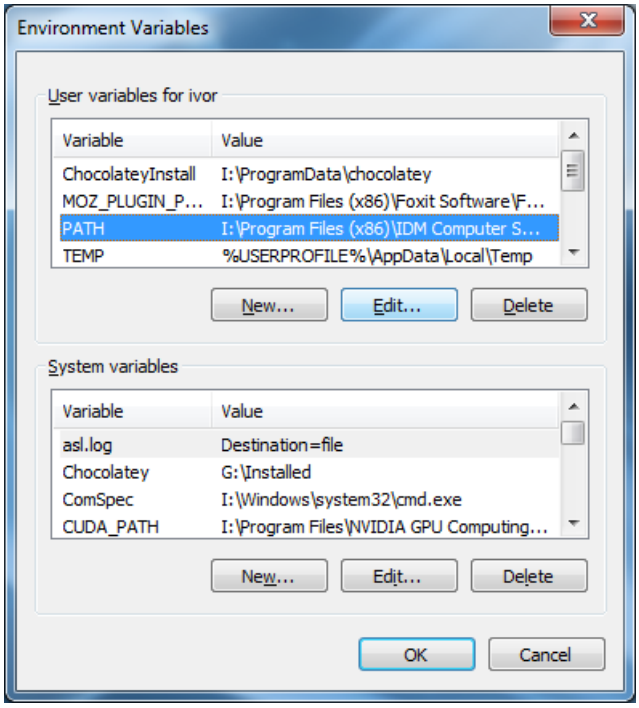
Info

- This tutorial shows how to edit PATH Environment Variable by appending directory where geth.exe is installed.
- This will allow you to execute geth from any directory.
- Without this step you would first need to position yourself in directory where GETH is installed by calling `cd I:\ProgramData\chocolatey\lib\geth-stable\tools` before being able to run GETH inside Command Prompt.

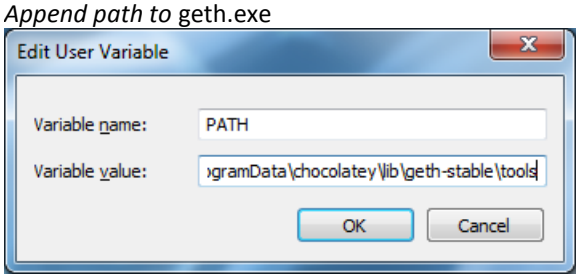
Procedure

- Show Environment Variables
- User Variables
- PATH
- Edit
- Variable value: **(Append ;I:\ProgramData\chocolatey\lib\geth-stable\tools)**
- 3*OK

Edit PATH

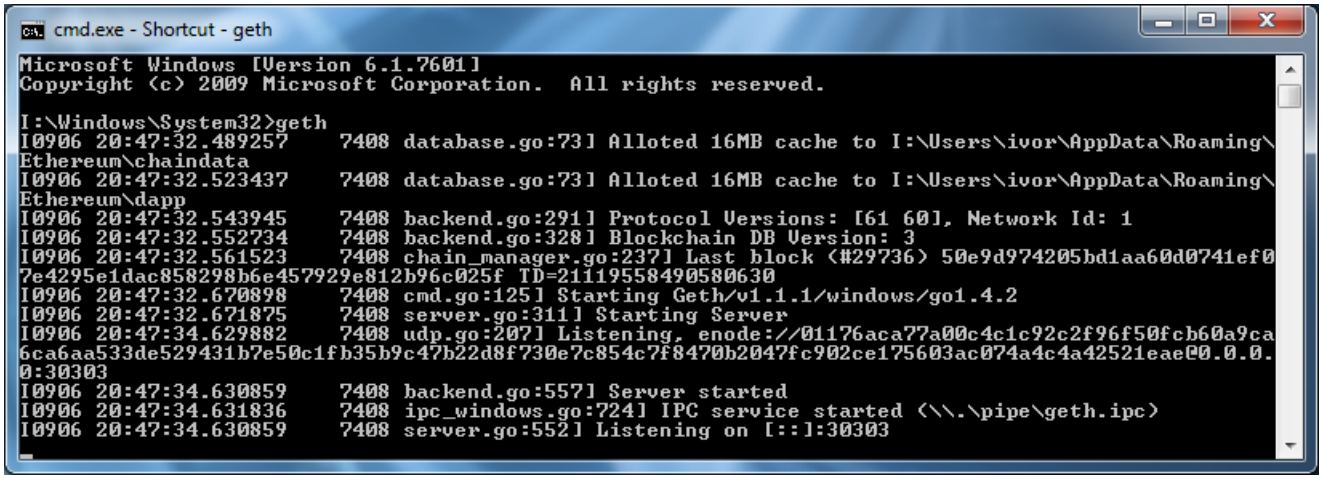


Append path to geth.exe



Test

- Start Command Prompt
- `geth`



2.1.2 Private Network

Info

- Following tutorials show how to use GETH to create and work with Private Network.

2.1.2.1 Setup

Info [R] [R]

- This tutorial shows how to setup Private Network for testing by adding initial amount of Ether to the Account.
- After that you can create additional accounts and mine more Ether.
- This way you can test creating and using contracts for which Ether is needed.

Initialize Network

- Create directory G:\Temp\Ethereum\datadir
- Create file G:\Temp\Ethereum\genesis_block2.json as shown below
- Start Command Prompt
- (Execute below commands)
- Remove all folders except keystore from G:\Temp\Ethereum\datadir

genesis_block2.json

```
{
  "nonce"      : "0xdeadbeefdeadbeef",
  "timestamp"  : "0x0",
  "parentHash" : "0x0000000000000000000000000000000000000000000000000000000000000000",
  "extraData"  : "0x0",
  "gasLimit"   : "0x8000000",
  "difficulty" : "0x400",
  "mixhash"    : "0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase"   : "0x3333333333333333333333333333333333333333",
  "alloc"      : {
  }
}
```

Start Private Network

```
geth --genesis G:\Temp\Ethereum\genesis_block2.json --datadir G:\Temp\Ethereum\datadir --networkid 123 --
nodiscover --maxpeers 0 console
```

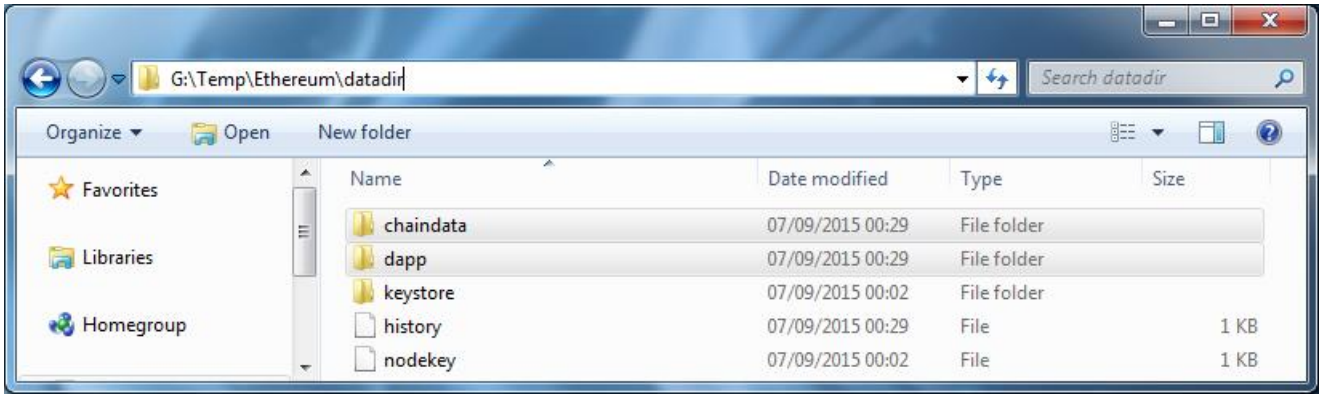
Create Account

```
> personal.newAccount("mypassword");
"0xae442df5214070f2eb98f3cfb9c3177918c2941a"
```

Exit Network

```
> exit
```

Remove all folders except keystore



Add Ether to the Account

- Replace G:\Temp\Ethereum\genesis_block2.json with file shown below using address of created Account
- (Execute below commands)

genesis_block2.json

```
{
  "nonce"      : "0xdeadbeefdeadbeef",
  "timestamp"  : "0x0",
  "parentHash" : "0x0000000000000000000000000000000000000000000000000000000000000000",
  "extraData"  : "0x0",
  "gasLimit"   : "0x8000000",
  "difficulty" : "0x400",
  "mixhash"    : "0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase"   : "0x3333333333333333333333333333333333333333",
  "alloc"      : {
     "0xae442df5214070f2eb98f3cfb9c3177918c2941a": {
        "balance": "1000000000000000000"
     }
  }
}
```

Start Private Network

```
geth --genesis G:\Temp\Ethereum\genesis_block2.json --datadir G:\Temp\Ethereum\datadir --networkid 123 --
nodiscover --maxpeers 0 console
```

Get Account balance

```
> web3.fromWei(eth.getBalance(eth.accounts[0]), "ether");
1000000000000000000
```

2.1.2.2 Connect

Info [R]

- This tutorial shows how to connect to Private Network which was setup using Setup Private Network.
- We are using GETH command option `2>> G:\Temp\my.log` to separate your commands from the log lines.
- Or you can use console command `geth attach` to open new Command Prompt where log lines will not be displayed.

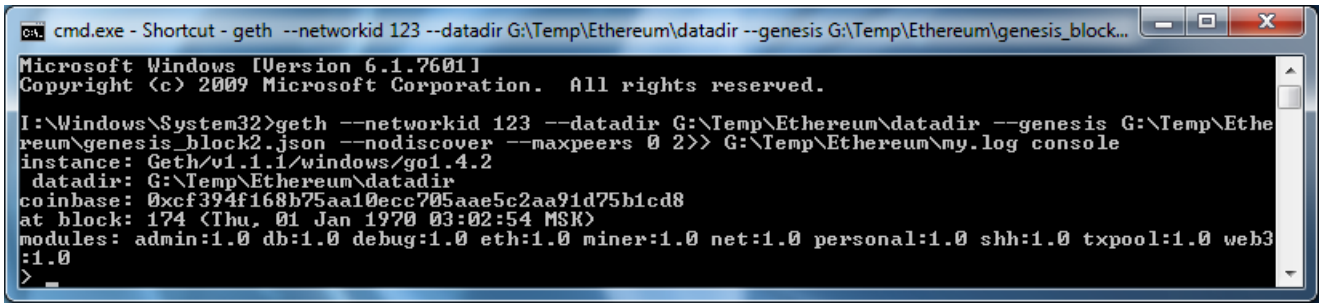
Procedure

- Start Command Prompt
- (Execute below commands)

Connect to private network

```
geth --networkid 123 --datadir G:\Temp\Ethereum\datadir --genesis G:\Temp\Ethereum\genesis_block2.json --
nodiscover --maxpeers 0 --solc "I:/Installed/Programming/Ethereum 0.9.41/Release/solc.exe" console 2>>
G:\Temp\Ethereum\my.log
```

Command Prompt



G:\Temp\Ethereum\my.log

```
I0909 11:34:55.546875    4356 database.go:73] Alloted 16MB cache to G:\Temp\Ethereum\datadir\chaindata
I0909 11:34:55.588867    4356 database.go:73] Alloted 16MB cache to G:\Temp\Ethereum\datadir\dapp
I0909 11:34:55.617187    4356 backend.go:291] Protocol Versions: [61 60], Network Id: 123
I0909 11:34:55.619140    4356 genesis.go:88] Genesis block already in chain. Writing canonical number
I0909 11:34:55.619140    4356 backend.go:303] Successfully wrote genesis block. New genesis hash =
378266f167238c543183a5230b251c50d431aadee179cedb937f6df8d4b707c9
I0909 11:34:55.620117    4356 backend.go:328] Blockchain DB Version: 3
I0909 11:34:55.620117    4356 chain_manager.go:237] Last block (#174)
c89e03b0a6a60cc9aa85a56c33f3d018c25c3c8b3912a0100ce576a82dc595fc TD=23791184
I0909 11:34:55.653320    4356 cmd.go:125] Starting Geth/v1.1.1/windows/go1.4.2
I0909 11:34:55.653320    4356 server.go:311] Starting Server
I0909 11:34:55.654296    4356 backend.go:557] Server started
I0909 11:34:55.654296    4356 ipc_windows.go:724] IPC service started (\\.\pipe\geth.ipc)
I0909 11:34:55.654296    4356 server.go:552] Listening on [::]:30303
```

2.1.3 Account

Info

- Following tutorials show how to use GETH to work with Accounts.

2.1.3.1 Create

Info [R]

- This tutorial shows how to use GETH interactive interface to create new Account.
- You will be required to type a password which needs to be entered twice.
- Command will return address which is assigned to the account.

Procedure

- Start Command Prompt
- (Execute below commands)

Commands

```
geth account new
Passphrase: B...5
Repeat Passphrase: B...5
```

Command Prompt

2.1.3.2 Unlock

Info [R]

- This tutorial shows how to use GETH to unlock Account on the Private Network.
- This will connect you to the network and ask for account password to unlock it.
- When using Public Network simply remove `--datadir G:\Temp\Ethereum\datadir`.

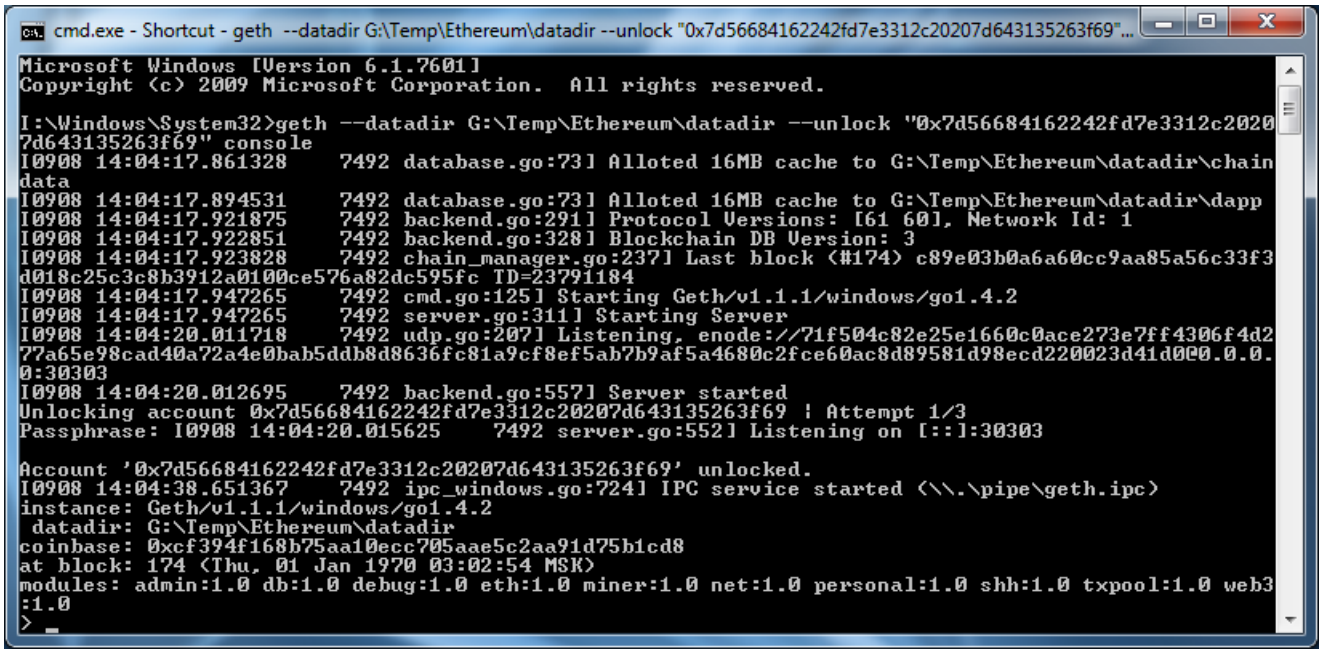
Procedure

- Start Command Prompt
- (Execute below commands)

Commands

```
geth --datadir G:\Temp\Ethereum\datadir --unlock "0x7d56684162242fd7e3312c20207d643135263f69" console
Passphrase: B...5
```

Command Prompt



2.1.3.3 Change Password

Info [R]

- This tutorial shows how to use GETH interactive interface to change account password by specifying its address.
- You will be asked to give the current password and then to type the new password twice.

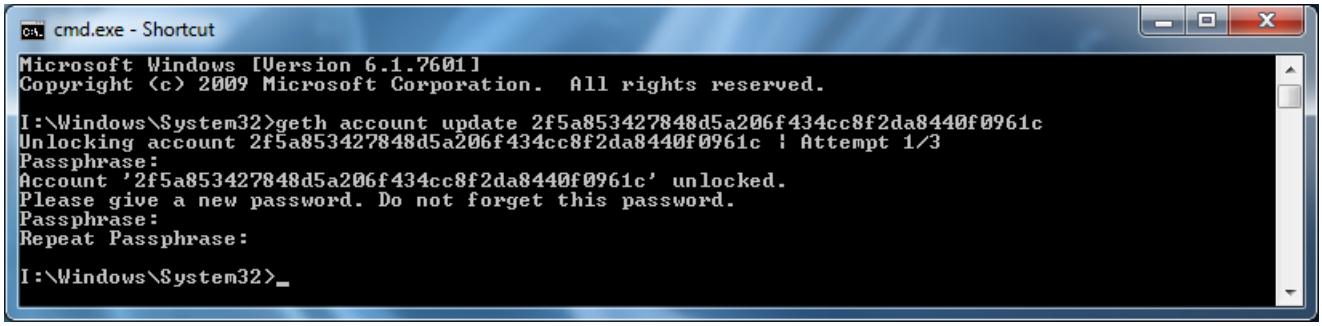
Procedure

- Start Command Prompt
- (Execute below commands)

Commands

```
geth account update 2f5a853427848d5a206f434cc8f2da8440f0961c
Passphrase: (old password)
Passphrase: (new password)
Repeat Passphrase: (new password)
```

Command Prompt



2.1.3.4 Show All

Info [R]

- This tutorial shows how to use GETH interactive interface to list accounts by displaying their indexes and addresses.

Procedure

- Start Command Prompt
- (Execute below commands)

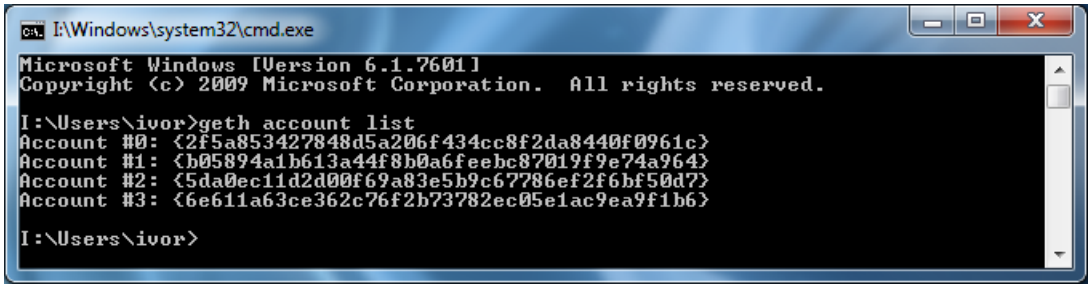
Command for Public Network

```
geth account list
```

Command for Private Network

```
geth --datadir G:\Temp\Ethereum\datadir account list
```

Command Prompt



2.1.3.5 Delete

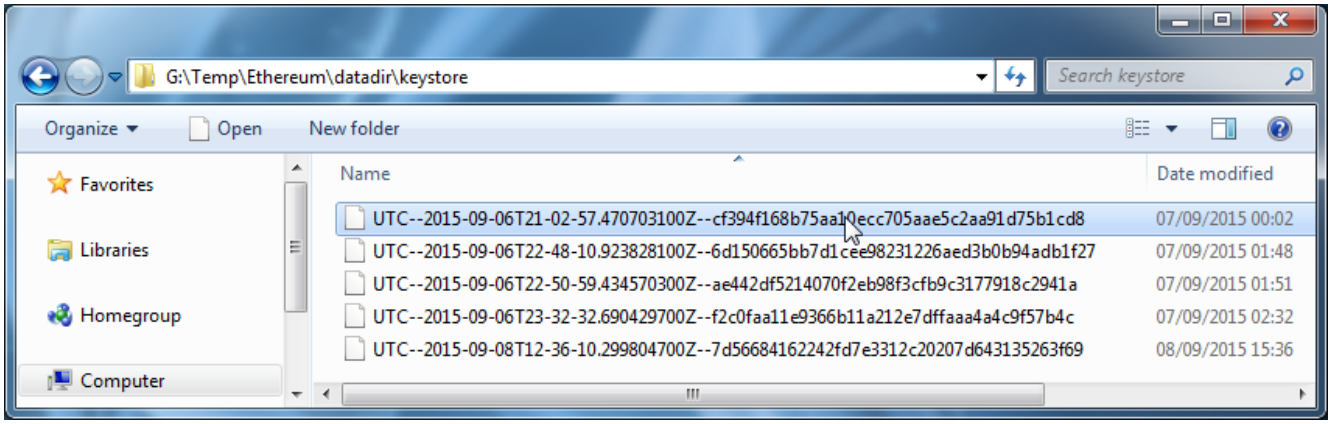
Info [R]

- This tutorial shows how to delete Account by removing its corresponding file from the keystore directory inside datadir.
- Account can't be deleted through GETH Command Line Parameters or Console.

Procedure

- Open directory G:\Temp\Ethereum\datadir\keystore
- Delete Account file

Windows Explorer



UTC--2015-09-06T21-02-57.470703100Z--cf394f168b75aa10ecc705aae5c2aa91d75b1cd8


```
{
  "address":"cf394f168b75aa10ecc705aae5c2aa91d75b1cd8",
  "Crypto":{
    "cipher":"aes-128-ctr",
    "ciphertext":"3106f829e8d3145127269a0976286147a8f82cb5bf43d2c5bcae65c689129b09",
    "cipherparams":{
      "iv":"6a9c34e1e41307dd6b9f29c359e1bc24"
    },
    "kdf":"scrypt",
    "kdfparams":{
      "dklen":32,
      "n":262144,
      "p":1,
      "r":8,
      "salt":"89960487c6b1b426d0f6e9735387edc6dc413c73007f7da1971ff3d779ff14b1"
    },
    "mac":"21db55bc4018b32c005822e7ab5c8eef45ae708bb47e3a6c8745c19ea36d1ae9"
  },
  "id":"aa6584e4-65b1-44ec-854b-c0b5ce5b3aa4",
  "version":3
}
```

2.1.3.6 Set Base Account

Info

- This tutorial shows how to set base Account which defines where Mined Ether should go.

Procedure

- Connect to Private Network
- (Execute below commands)

Set base account.

```
miner.setEtherbase(eth.accounts[0]);
```

2.1.3.7 Get balance

Info

- This tutorial shows how to get amount of Ether stored into an Account.

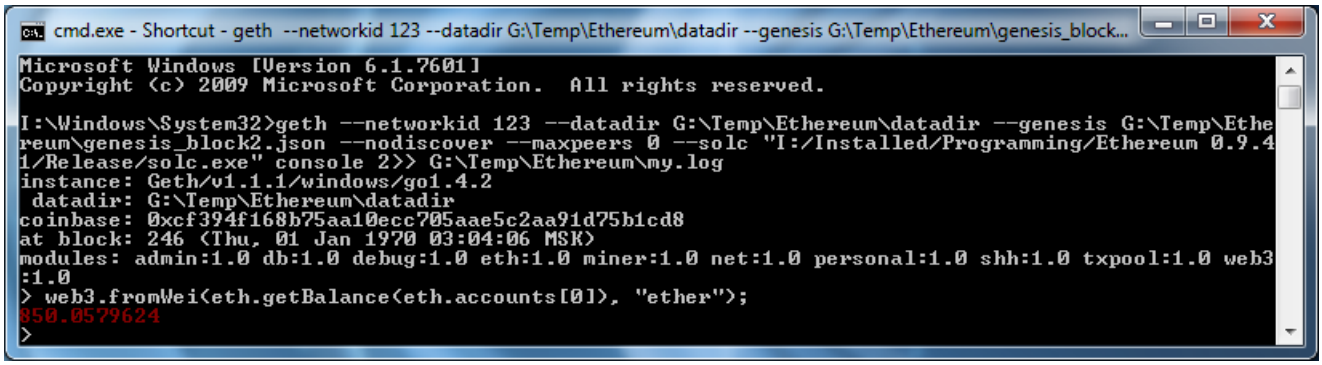
Procedure

- Connect to Private Network
- (Execute below commands)

Get balance

```
web3.fromWei(eth.getBalance(eth.accounts[0]), "ether");
```

Geth



2.1.4 Ether

Info

- Following tutorials show how to use GETH with Solidity Compiler (SOLC).

2.1.4.1 Mine

Info

- This tutorial shows how to mine Ether on Private Network.
- When new account is created, in order to get its index, you might need to `exit` console and then use

  `geth --datadir G:\Temp\Ethereum\datadir account list`

Procedure

- Connect to Private Network
- (Execute below commands)

  Create additional Account

  ```
  > personal.newAccount("mypassword");
  ```

  Set base account so that mined Ether is added to its balance

  ```
  > miner.setEtherbase(eth.accounts[0]);
  ```

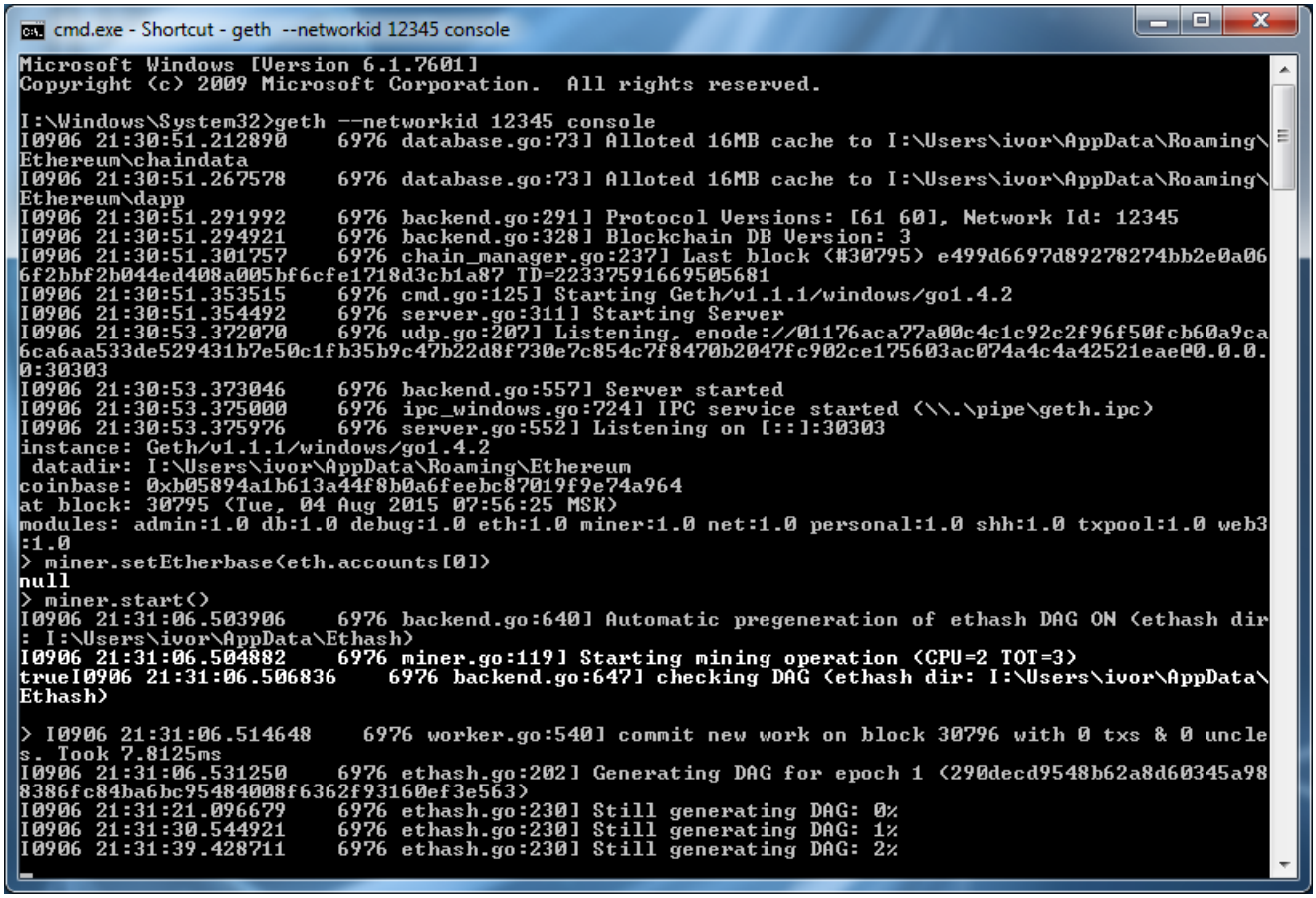
  Start mining. Wait for creation of DAG file if one doesn't exist.

  ```
  > miner.start();
  ```

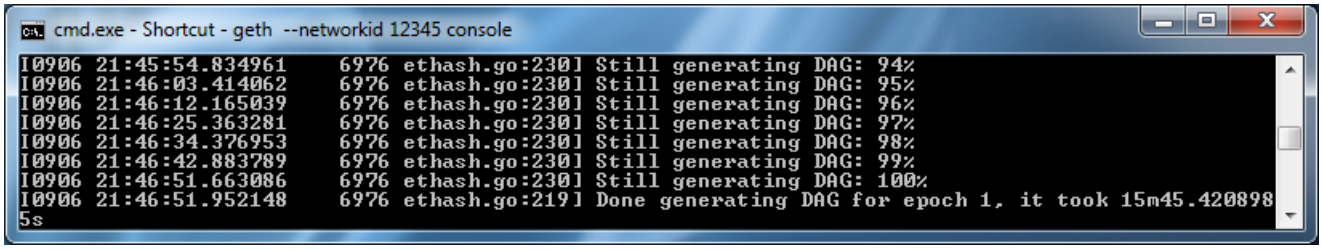
  Check balance of the base account

  ```
  > web3.fromWei(eth.getBalance(eth.accounts[0]), "ether");
  10
  ```

  Creating DAG file


cmd.exe - Shortcut - geth --networkid 12345 console
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
I:\Windows\System32>geth --networkid 12345 console
I0906 21:30:51.212890 6976 database.go:73] Alloted 16MB cache to I:\Users\ivor\AppData\Roaming\Ethereum\chaindata
I0906 21:30:51.267578 6976 database.go:73] Alloted 16MB cache to I:\Users\ivor\AppData\Roaming\Ethereum\dapp
I0906 21:30:51.291992 6976 backend.go:291] Protocol Versions: [61 60], Network Id: 12345
I0906 21:30:51.294921 6976 backend.go:328] Blockchain DB Version: 3
I0906 21:30:51.301757 6976 chain_manager.go:237] Last block (#30795) e499d6697d89278274bb2e0a066f2bbf2b044ed408a005bf6cfe1718d3cb1a87 TD=22337591669505681
I0906 21:30:51.353515 6976 cmd.go:125] Starting Geth/v1.1.1/windows/go1.4.2
I0906 21:30:51.354492 6976 server.go:311] Starting Server
I0906 21:30:53.372070 6976 udp.go:207] Listening, enode://01176aca77a00c4c1c92c2f96f50fcb60a9ca6ca6aa533de529431b7e50c1fb35b9c47b22d8f730e7c854c7f8470b2047fc902ce175603ac074a4c4a42521eae@0.0.0.0:30303
I0906 21:30:53.373046 6976 backend.go:557] Server started
I0906 21:30:53.375000 6976 ipc_windows.go:724] IPC service started (\\.\pipe\geth.ipc)
I0906 21:30:53.375976 6976 server.go:552] Listening on [::]:30303
instance: Geth/v1.1.1/windows/go1.4.2
datadir: I:\Users\ivor\AppData\Roaming\Ethereum
coinbase: 0xb05894a1b613a44f8b0a6feebc87019f9e74a964
at block: 30795 (Tue, 04 Aug 2015 07:56:25 MSK)
modules: admin:1.0 db:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 shh:1.0 txpool:1.0 web3:1.0
> miner.setEtherbase(eth.accounts[0])
null
> miner.start()
I0906 21:31:06.503906 6976 backend.go:640] Automatic pregeneration of ethash DAG ON (ethash dir: I:\Users\ivor\AppData\Ethash)
I0906 21:31:06.504882 6976 miner.go:119] Starting mining operation (CPU=2 TOT=3)
trueI0906 21:31:06.506836 6976 backend.go:647] checking DAG (ethash dir: I:\Users\ivor\AppData\Ethash)
> I0906 21:31:06.514648 6976 worker.go:540] commit new work on block 30796 with 0 txs & 0 uncles. Took 7.8125ms
I0906 21:31:06.531250 6976 ethash.go:202] Generating DAG for epoch 1 (290decd9548b62a8d60345a988386fc84ba6bc95484008f6362f93160ef3e563)
I0906 21:31:21.096679 6976 ethash.go:230] Still generating DAG: 0%
I0906 21:31:30.544921 6976 ethash.go:230] Still generating DAG: 1%
I0906 21:31:39.428711 6976 ethash.go:230] Still generating DAG: 2%

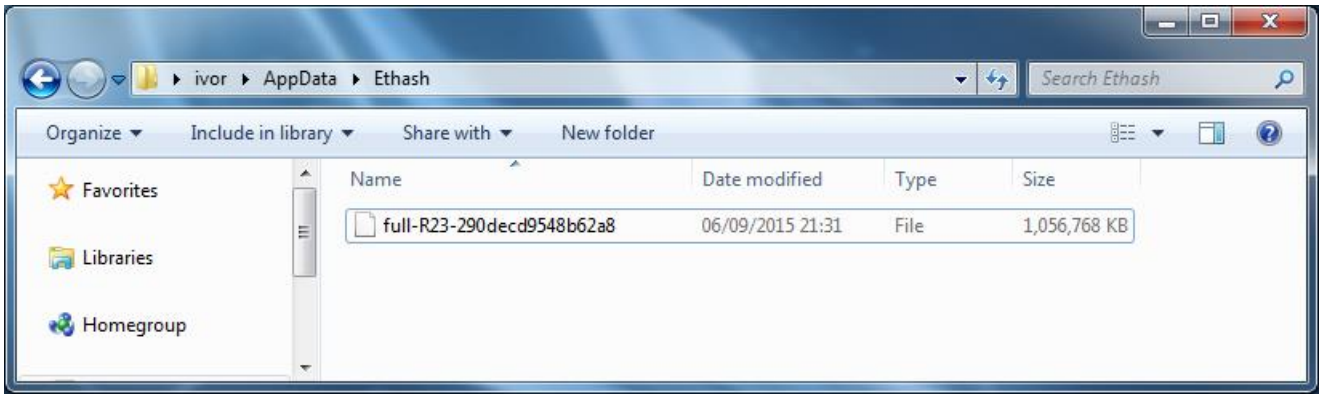
DAG file created



Mining

Check balance

Created DAG file



2.1.4.2 Transfer

Info [R]

- This tutorial shows how to transfer Ether between Accounts.
- Before transferring Ether we will execute GETH command which list all Accounts with their indexes and addresses since we will need indexes to check balances and addresses to transfer Ether.

List Accounts

- Start Command Prompt
- (Execute below commands)

List Accounts

```
> geth --datadir G:\Temp\Ethereum\datadir account list
Account #0: {cf394f168b75aa10ecc705aae5c2aa91d75b1cd8}
Account #1: {6d150665bb7d1cee98231226aed3b0b94adb1f27}
Account #2: {ae442df5214070f2eb98f3cfb9c3177918c2941a}
Account #3: {f2c0faa11e9366b11a212e7dffaaa4a4c9f57b4c}
Account #4: {7d56684162242fd7e3312c20207d643135263f69}
```

Connect to private network

```
geth --networkid 123 --datadir G:\Temp\Ethereum\datadir --genesis G:\Temp\Ethereum\genesis_block2.json --
nodiscover --maxpeers 0 --solc "I:/Installed/Programming/Ethereum 0.9.41/Release/solc.exe" console
```

Get Ether balances from source and destination

```
web3.fromWei(eth.getBalance(eth.accounts[4]), "ether");
web3.fromWei(eth.getBalance(eth.accounts[0]), "ether");
```

Transfer Ether

```
eth.sendTransaction({from: '0x7d56684162242fd7e3312c20207d643135263f69', to:
'0xcf394f168b75aa10ecc705aae5c2aa91d75b1cd8', value: web3.toWei(10, "ether")})
```

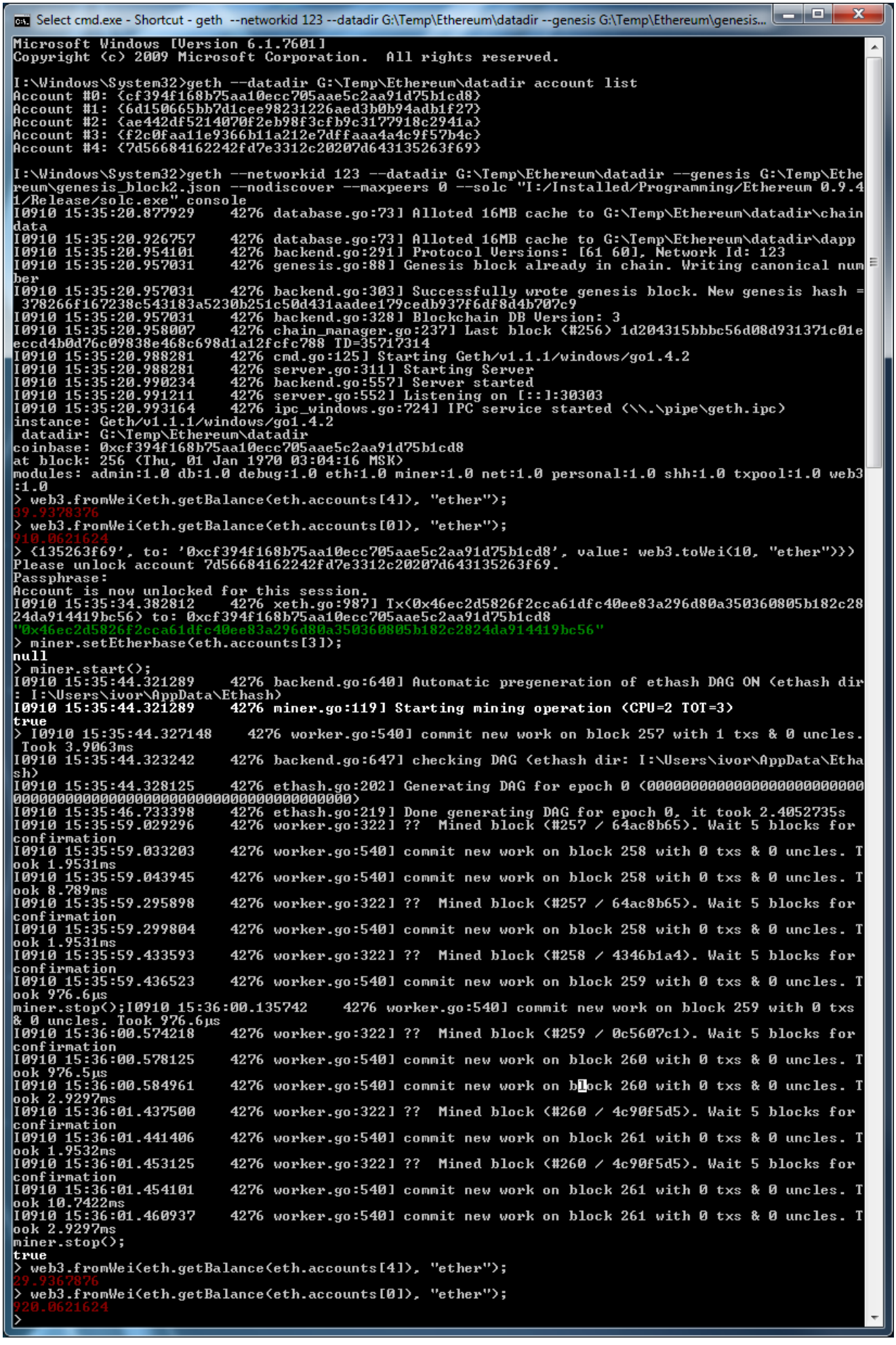
Set base account. Start mining. Wait a while. Stop mining.

```
miner.setEtherbase(eth.accounts[3]);
miner.start();
miner.stop();
```

Get Ether balances from source and destination

```
web3.fromWei(eth.getBalance(eth.accounts[4]), "ether");
web3.fromWei(eth.getBalance(eth.accounts[0]), "ether");
```

Command Prompt



2.1.5 SOLC

Info

- Following tutorials show how to link GETH with installed Solidity Compiler (SOLC).
- You don't need to install Solidity Compiler (SOLC) if you plan on using some of the online compilers as described in
  - Create contract using online SOLC - Browser Solidity
  - Create contract using online SOLC - Ether Chain

2.1.5.1 Check if GETH is linked to SOLC

Info

- This tutorial shows how to check if GETH is linked to SOLC.

Procedure

- Start Command Prompt
- (Execute below commands)

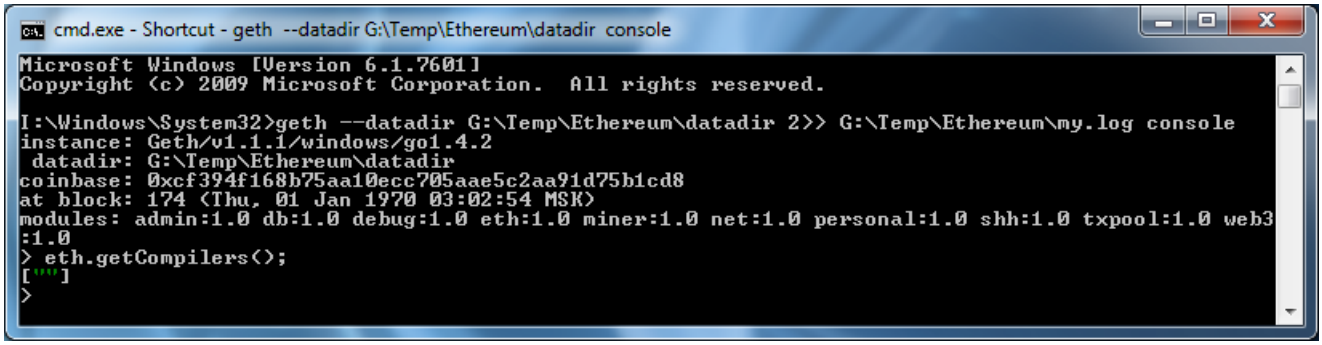
Connect to private network

```
geth --datadir G:\Temp\Ethereum\datadir 2>> G:\Temp\Ethereum\my.log console
```

Test if GETH is linked to SOLC

```
> eth.getCompilers();
["Solidity"]
```

Command Prompt



2.1.5.2 Link GETH with SOLC

Info [R]

- This tutorial shows how to link GETH with SOLC.
- Prerequisite is to install Solidity Compiler by Installing ETH which includes solc.exe.
- You have to repeat this procedure every time you reconnect to the network.

Using GETH Command Line

- Start Command Prompt
- (Execute below commands)

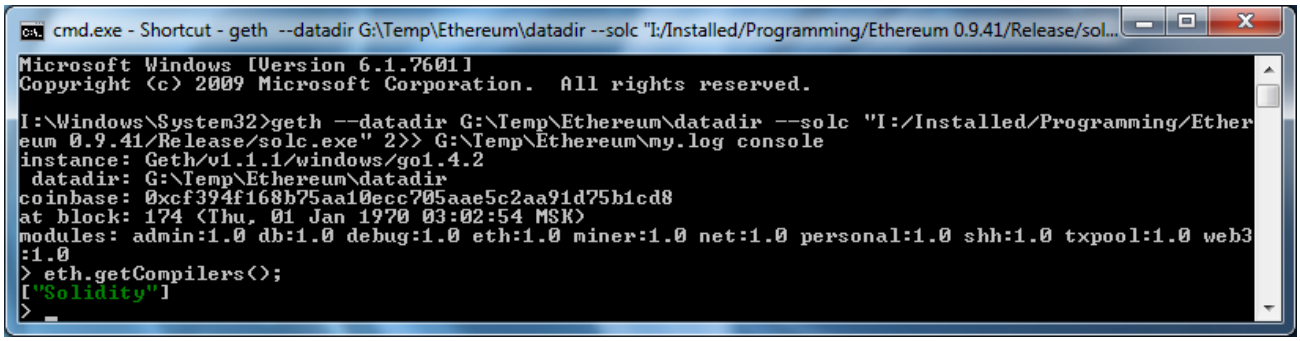
Connect to private network

```
geth --datadir G:\Temp\Ethereum\datadir --solc "I:/Installed/Programming/Ethereum 0.9.41/Release/solc.exe" 2>>
G:\Temp\Ethereum\my.log console
```

Test if linking was successful

```
> eth.getCompilers();
["Solidity"]
```

Command Prompt



Using GETH Console

- Start Command Prompt
- (Execute below commands)

Connect to private network

```
geth --datadir G:\Temp\Ethereum\datadir 2>> G:\Temp\Ethereum\my.log console
```

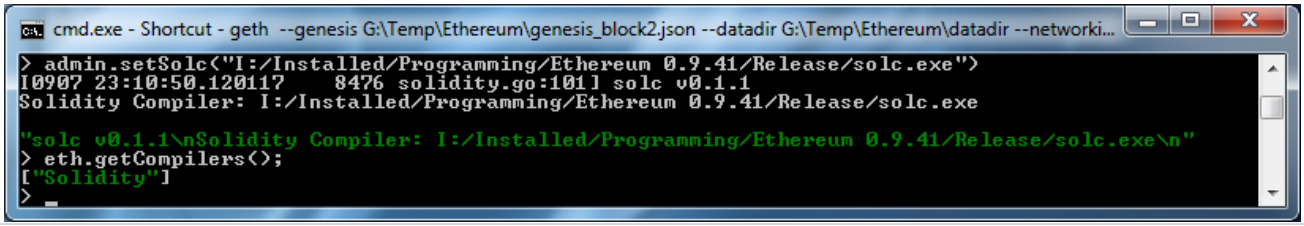
Link GETH with SOLC

```
> admin.setSolc("I:/Installed/Programming/Ethereum 0.9.41/Release/solc.exe");
"solc v0.1.1\nSolidity Compiler: I:/Installed/Programming/Ethereum 0.9.41/Release/solc.exe\n"
```

Test if linking was successful

```
> eth.getCompilers();
["Solidity"]
```

Command Prompt



2.1.6 Contracts

Info

- Following tutorials show how to work with contracts.
- Contracts can be created either by using installed or online Solidity Compilers as shown by the tutorials.

2.1.6.1 Create using online SOLC - Browser Solidity

Info

- This tutorial shows how to create contract using online Solidity Compiler Browser Solidity.
- It creates deployment code "Web3 deploy" in which values for input variables should be inserted before pasting to GETH.
- Compiler also creates Bytecode and Contract ABI which can be used as shown in Create using online SOLC - Ether Chain.

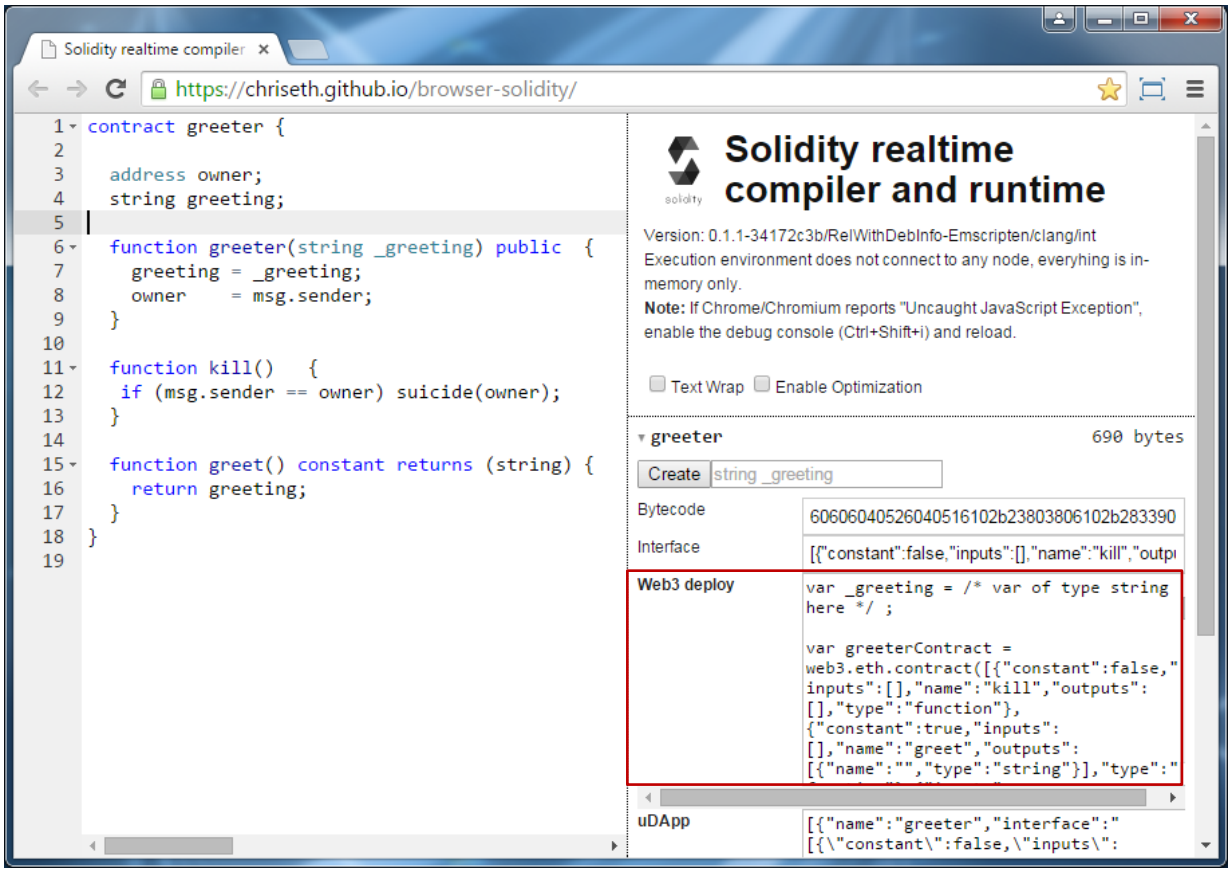
Compile Code

- https://chriseth.github.io/browser-solidity/
- (Paste Contract Source Code)

Contract Source Code

```
contract greeter {

  address owner;                                          /* Define address variable owner   */
  string  greeting;                                       /* Define string variable greeting */

  function greeter(string _greeting) public  {            /* Run during initialization       */
    greeting = _greeting;
    owner    = msg.sender;
  }

  function kill()   {                                     /* Get funds on killing contract  */
   if (msg.sender == owner) suicide(owner);
  }

  function greet() constant returns (string) {            /* Main function                  */
    return greeting;
  }

}
```

https://chriseth.github.io/browser-solidity/



Create Contract

- Connect to Private Network
- (Paste "Web3 deploy" into text editor)
- (Enter values for input variables `var _greeting = /* var of type string here */ ;`)
- (Execute below commands)

Unlock Account

```
personal.unlockAccount("7d56684162242fd7e3312c20207d643135263f69", "mypassword")
```

Paste "Web3 deploy" with entered values for input variables

```
var _greeting = "Hello World!" ;

var greeterContract =
web3.eth.contract([{"constant":false,"inputs":[],"name":"kill","outputs":[],"type":"function"},{"constant":true
,"inputs":[],"name":"greet","outputs":[{"name":"","type":"string"}],"type":"function"},{"inputs":[{"name":"_gre
eting","type":"string"}],"type":"constructor"}]);
var greeter = greeterContract.new(
   _greeting,
   {
     from: web3.eth.accounts[4],
     data:
'6060604052604051610 2b23803806102b2833901604052606080516060019060200150 5b806001600050908051906020019082805482 8
5590600052602060002090601f016020900481019282156071579182015b828111156070578251826000505591602001919060010190605
4565b5b509050609891906 07c565b8082111560945760008181506000905550600101607c565b5090565b5050336000600061010 00a8154
8173ffffffffffffffffffffffffffffffffffffffff021916908302179055505b506101dc806100d66000396000f300606060405260003
57c0100000000000000000000000000000000000000000000000000000000009004806341c0e1b514610044578063cfae3217146100515761
0042565b005b61004f6004506100ca565b005b61005c60045061015e565b604051808060200182810382528381815181526020019150805
1906020019080838382906000600460208460 1f0104600302600f01f150905090810190601f1680156100bc57808203805160018360020 03
6101000a031916815260200191505b5092505050604051809103 90f35b600060009054906101000a900473ffffffffffffffffffffffff
ffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff161415
61015b576000600090549061010 00a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffff
ffffffff16ff5b5b565b602060405190810160405280600081526020015060016000508054806 01f0160208091040260200160405190 81
016040528092919081815260200182805480156101cd57820191906000526020600020905b8154815290600101906020018083116101b05
7829003601f168201915b5050505090506101d9565b9056',
     gas: 1000000
   }, function(e, contract){
    if (typeof contract.address != 'undefined') {
         console.log(e, contract);
         console.log('Contract mined! address: ' + contract.address + ' transactionHash: ' +
contract.transactionHash);
    }
 })
```

Start mining. Wait for message "Contract mined!" Stop mining.

```
miner.start();
miner.stop();
```

Call Greeter. Errors: TypeError: 'greet' is not a function

```
greeter.greet();
```

Command Prompt

2.1.6.2 Create using online SOLC - Ether Chain

Info

- This tutorial shows how to create contract using online Solidity Compiler Ether Chain.
- Compiler will create Bytecode and Contract ABI which need to be saved in variables in GETH console as shown below.
- Compiled code needs to be saved in a variable using single or double quotes with or without leading 0x.

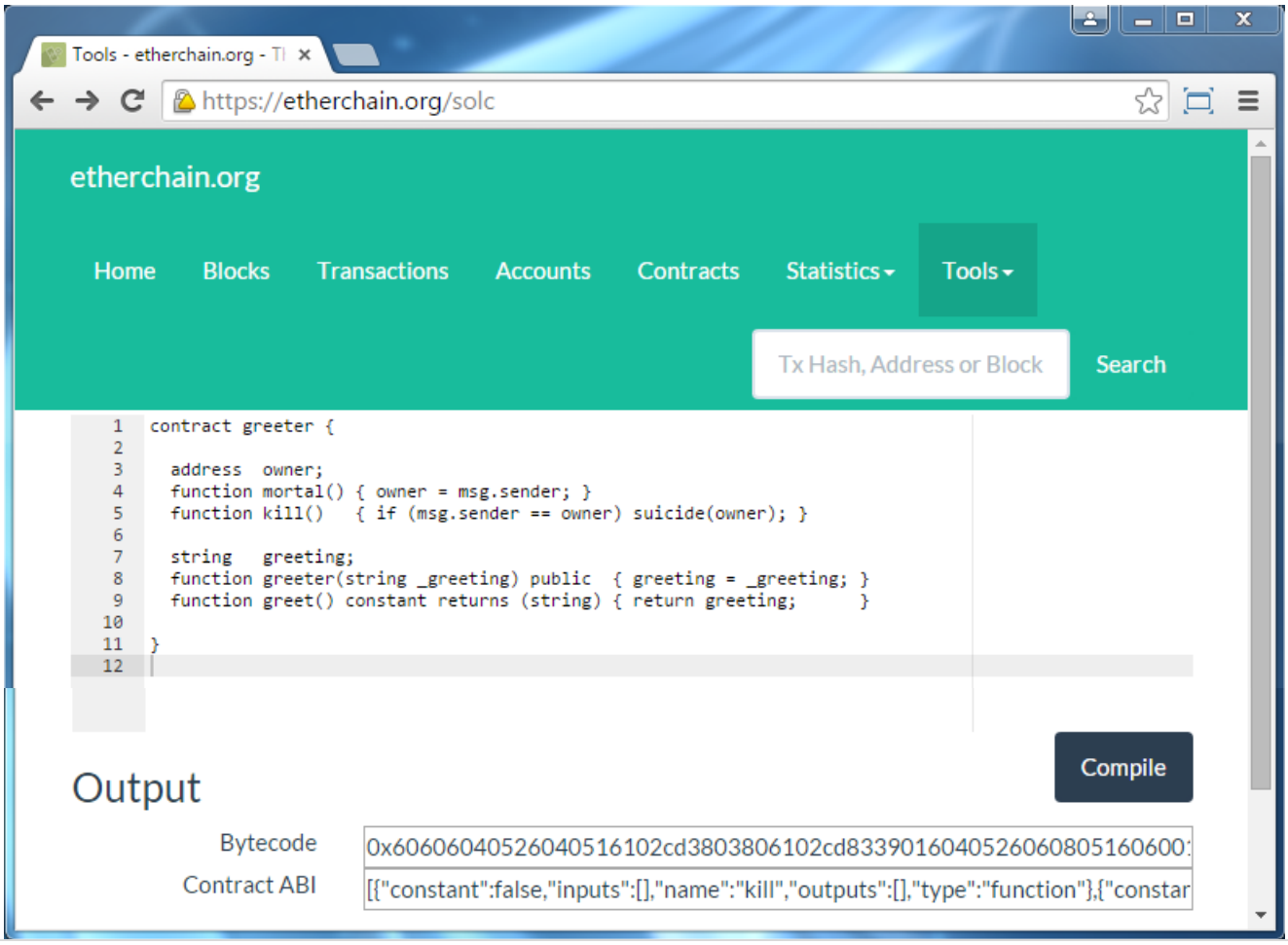
Compile Code

- https://etherchain.org/solc
- (Paste Contract Source Code)
- Compile

Contract Source Code

```
contract greeter {

  address owner;                                              /* Define address variable owner   */
  string greeting;                                            /* Define string variable greeting */

  function greeter(string _greeting) public  {                /* Run during initialization       */
    greeting = _greeting;
    owner    = msg.sender;
  }

  function kill()   { if (msg.sender == owner) suicide(owner); }      /* Get funds on killing contract  */

  function greet() constant returns (string) { return greeting; }     /* Main function                  */

}
```

Online Solidity Compiler



Create Contract

- Connect to Private Network
- (Execute below commands)

Copy Byte code into variable

```
var greeterCompiled =
'0x6060604052604051602b23803806102b2833901604052606080516060019602001505b806001600050908051906020019082805482
825590600052602060002090601f016020900481019282156071579182015b828111156070578251826000505591602001919060010190606
054565b5b5090506098919060 7c565b8082111560945760008181506000905550600101607c565b5090565b5050336000600061010 00a81
548173ffffffffffffffffffffffffffffffffffffffff02191690830217905550 5b506101dc806100d66000396000f3006060604052600
0357c0100000000000000000000000000000000000000000000000000000000009004806341c0e1b514610044578063cfae32171461005157
610042565b005b61004f6004506100ca565b005b61005c60045061015e565b604051808060200182810382528381815181526020019150 8
05190602001908083838290600060046020846 01f0104600302600f01f150905090810190601f1680156100bc57808203805160018360200
036101000a031916815260200191505b5092505050604051809103 90f35b600060009054906101000a900473ffffffffffffffffffffffff
ffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff1614
1561015b576000600090549061010 00a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffff
fffffffff16ff5b5b565b602060405190810160405280600081526020015060016000508054806 01f0160208091040260200160405190
810160405280929190818152602001828054801561 01cd57820191906000526020600020905b81548152906001019060200180831161 01b
057829003601f168201915b5050505050905061 01d9565b9056'
```

Copy Contract ABI into variable

```
var greeterABI =
[{"constant":false,"inputs":[],"name":"kill","outputs":[],"type":"function"},{"constant":true,"inputs":[],"name
":"greet","outputs":[{"name":"","type":"string"}],"type":"function"},{"inputs":[{"name":"_greeting","type":"str
ing"}],"type":"constructor"}]
```

Contract Object

```
var greeterContract = web3.eth.contract(greeterABI);
```

Prepare contract variable and source.

```
var _greeting       = "Hello World!"
```

Unlock Account

```
personal.unlockAccount("7d56684162242fd7e3312c20207d643135263f69", "mypassword")
```

Deploy Contract Asynchronously. You will get message "waiting to be mined..."

```
var greeter         = greeterContract.new(_greeting,{from:web3.eth.accounts[4], data: greeterCompiled, gas:
300000}, function(e, contract){
    if(!e) {
      if(!contract.address) {
        console.log("Contract transaction send: TransactionHash: " + contract.transactionHash + " waiting to be
mined...");
      } else {
        console.log("Contract mined! Address: " + contract.address);
        console.log(contract);
      }
    }
})
```

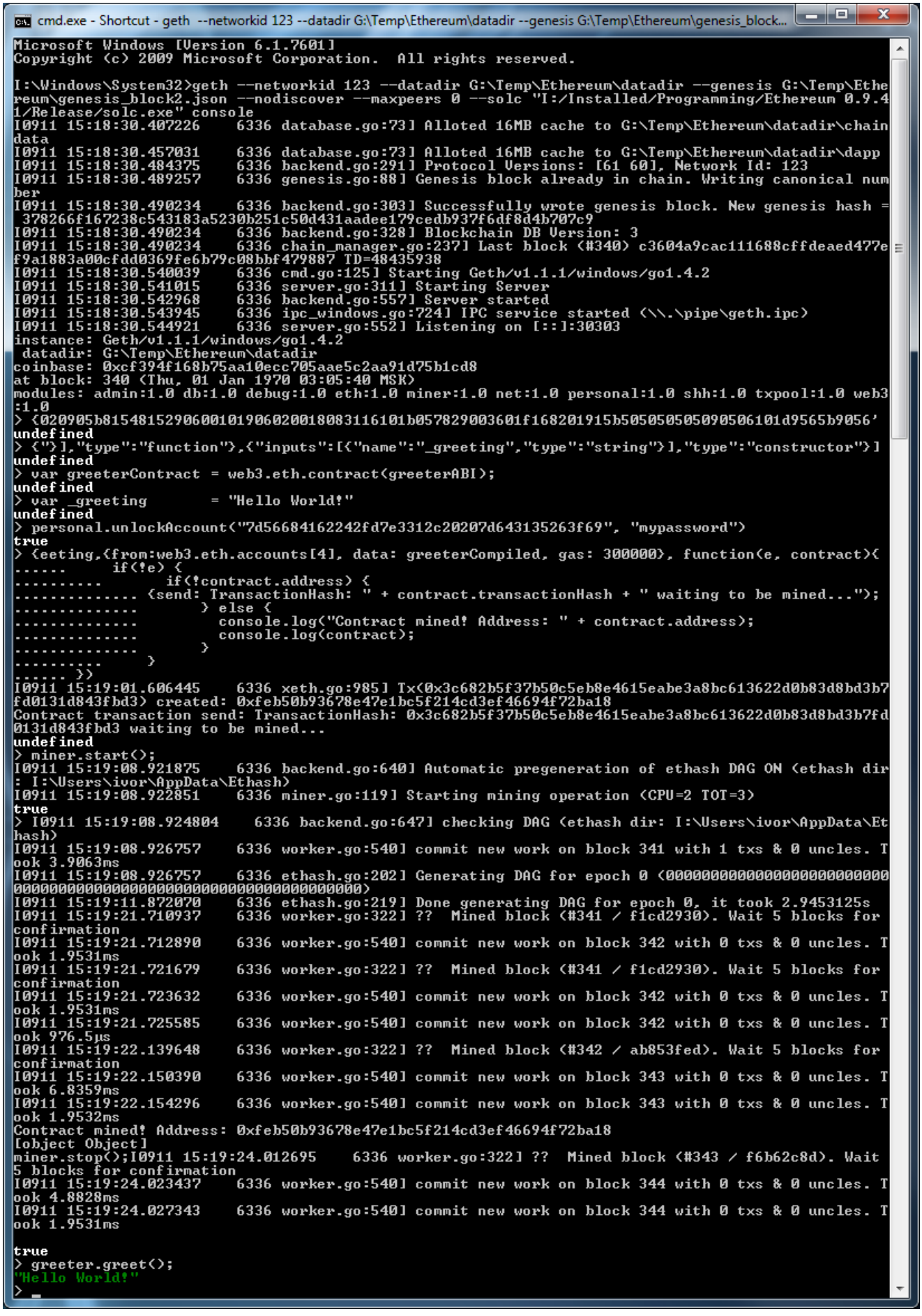
Start mining. Wait for message "Contract mined!" Stop mining.

```
miner.start();
miner.stop();
```

Call Greeter. Errors: TypeError: 'greet' is not a function

```
greeter.greet();
```

Command Prompt



2.1.6.3 Create using installed SOLC

Info **[R]**

- This tutorial shows how to create contract using installed SOLC and deploying it either Asynchronously or Synchronously.
- Tutorial uses Textfixer to remove line breaks from contract definition so that it can be stored in a variable.
- Example uses GETH command line parameter `--solc` to link with locally installed Solidity Compiler.

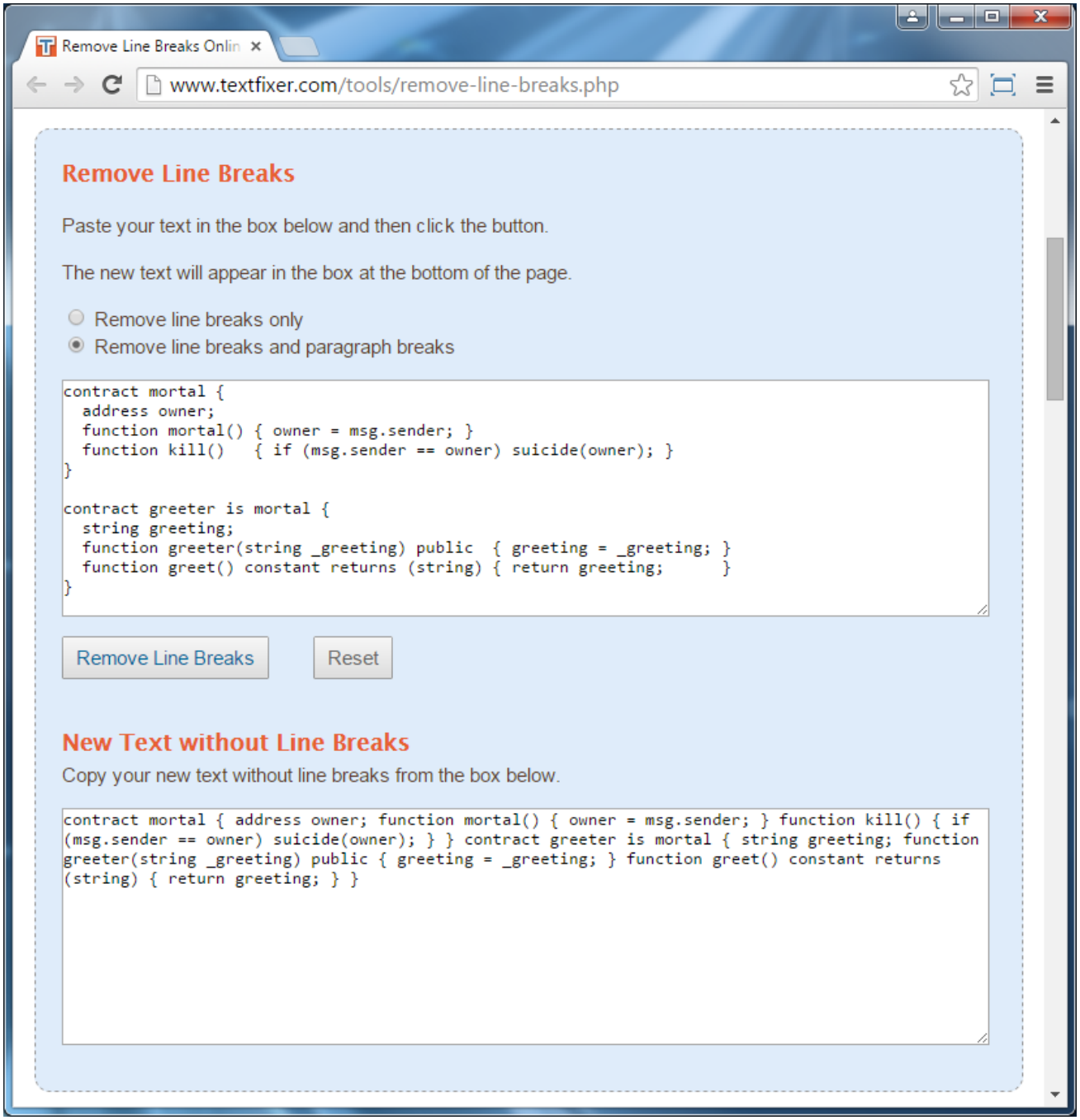
Remove line breaks

- http://www.textfixer.com/tools/remove-line-breaks.php
- Remove line breaks and paragraph breaks: SELECT
- (Paste contract definitions with or without comments)
- Remove Line Breaks

Contract definitions

```
contract mortal {
  address owner;                                                  /* Define address variable owner    */
  function mortal() { owner = msg.sender; }                       /* Set owner during initialization  */
  function kill()   { if (msg.sender == owner) suicide(owner); }  /* Get funds on killing contract    */
}

contract greeter is mortal {
  string greeting;                                                 /* Define string variable greeting */
  function greeter(string _greeting) public  { greeting = _greeting; }  /* Run during initialization       */
  function greet() constant returns (string) { return greeting;      }  /* Main function                   */
}
```

Text Fixer



Deploy Contract Asynchronously

- Connect to Private Network
- (Execute below commands)

Copy "New Text without Line Breaks" into variable

```
var greeterSource = 'contract mortal { address owner; function mortal() { owner = msg.sender; } function kill() { if (msg.sender == owner) suicide(owner); } } contract greeter is mortal { string greeting; function greeter(string _greeting) public { greeting = _greeting; } function greet() constant returns (string) { return greeting; } }';
```

Compile contracts

```
var greeterCompiled = web3.eth.compile.solidity(greeterSource);
```

Unlock Account

```
personal.unlockAccount("7d56684162242fd7e3312c20207d643135263f69", "mypassword")
```

Prepare contract variable and source.

```
var _greeting       = "Hello World!"
var greeterContract = web3.eth.contract(greeterCompiled.greeter.info.abiDefinition);
```

Deploy Contract Asynchronously. You will get message "waiting to be mined..."

```
var greeter  = greeterContract.new(_greeting,{from:web3.eth.accounts[4], data: greeterCompiled.greeter.code, gas: 300000}, function(e, contract){
    if(!e) {
      if(!contract.address) {
        console.log("Contract transaction send: TransactionHash: " + contract.transactionHash + " waiting to be mined...");
      } else {
        console.log("Contract mined! Address: " + contract.address);
        console.log(contract);
      }
    }
})
```

Start mining. Wait for message "Contract mined!" Stop mining.

```
miner.start();
miner.stop();
```

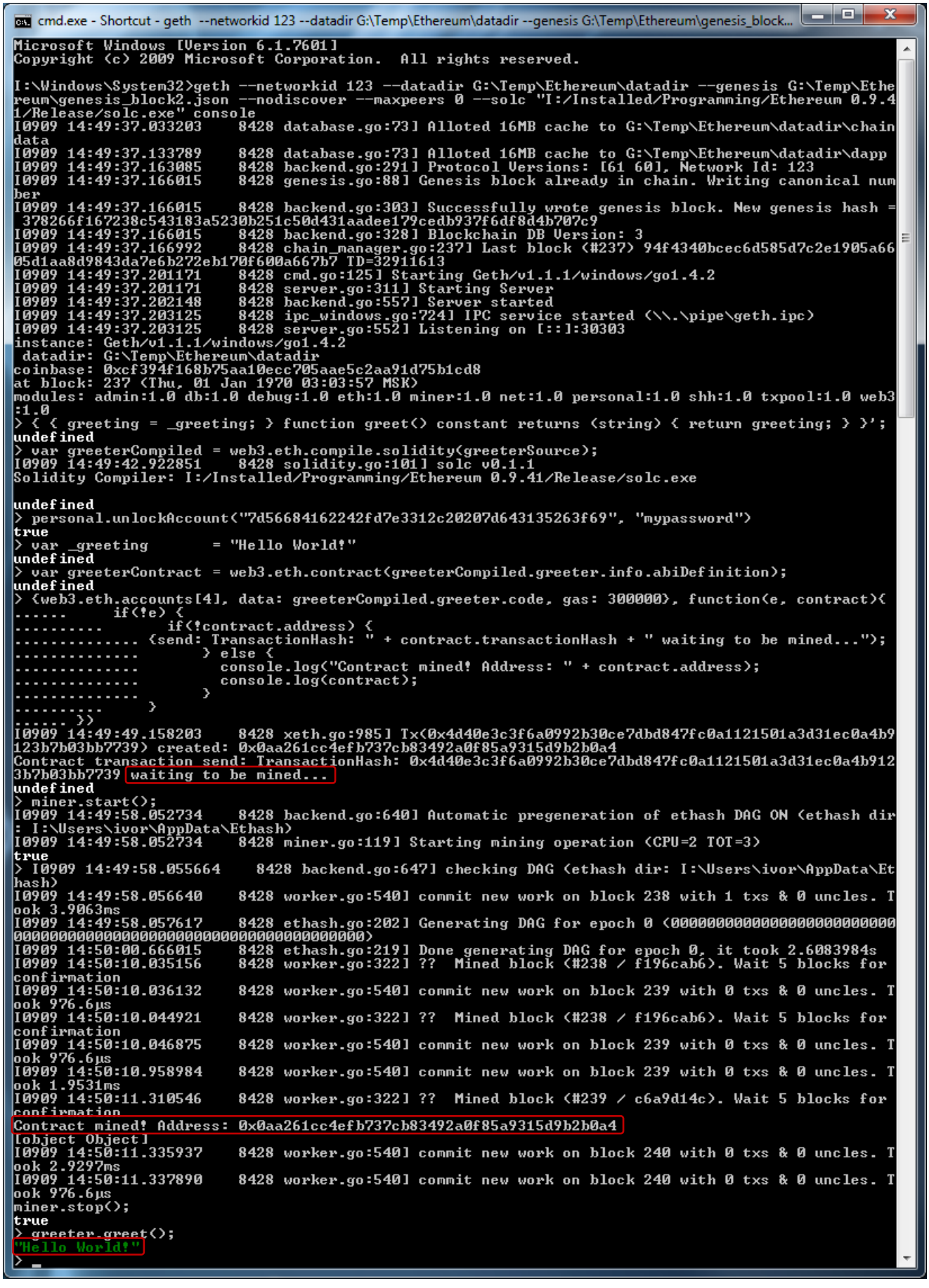
Call Greeter. Errors: TypeError: 'greet' is not a function

```
greeter.greet();
```

output

```
'Hello World!'
```

Command Prompt



Deploy Contract Synchronously [R]

- Connect to Private Network
- (Execute below commands)

Copy "New Text without Line Breaks" into variable

```
var greeterSource = 'contract mortal { address owner; function mortal() { owner = msg.sender; } function kill() { if (msg.sender == owner) suicide(owner); } } contract greeter is mortal { string greeting; function greeter(string _greeting) public { greeting = _greeting; } function greet() constant returns (string) { return greeting; } }';
```

Compile contracts

```
var greeterCompiled = web3.eth.compile.solidity(greeterSource);
```

Unlock Account

```
personal.unlockAccount("7d56684162242fd7e3312c20207d643135263f69", "mypassword")
```

Prepare contract variable and source.

```
var _greeting       = "Hello World!"
var greeterContract = web3.eth.contract(greeterCompiled.greeter.info.abiDefinition);
```

Deploy Contract Synchronously.

```
var greeter = greeterContract.new(_greeting, {from:web3.eth.accounts[4], data: greeterCompiled.greeter.code, gas: 300000});
```

Start mining. Wait a while. Stop mining.

```
miner.start();
miner.stop();
```

Call Greeter. Errors: TypeError: 'greet' is not a function

```
greeter.greet();
```

output

```
'Hello World!'
```

Command Prompt

2.1.6.4 Run by others

Info [R]

- This tutorial shows how to allow other people to run your Contract by using its
  - ABI (Application Binary Interface) which describes functions name and how to call them from the JavaScript
  - Address which defines where the contract is located
- You can send ABI and Address, shown at the end of tutorial, to anyone who wants to use your contract.

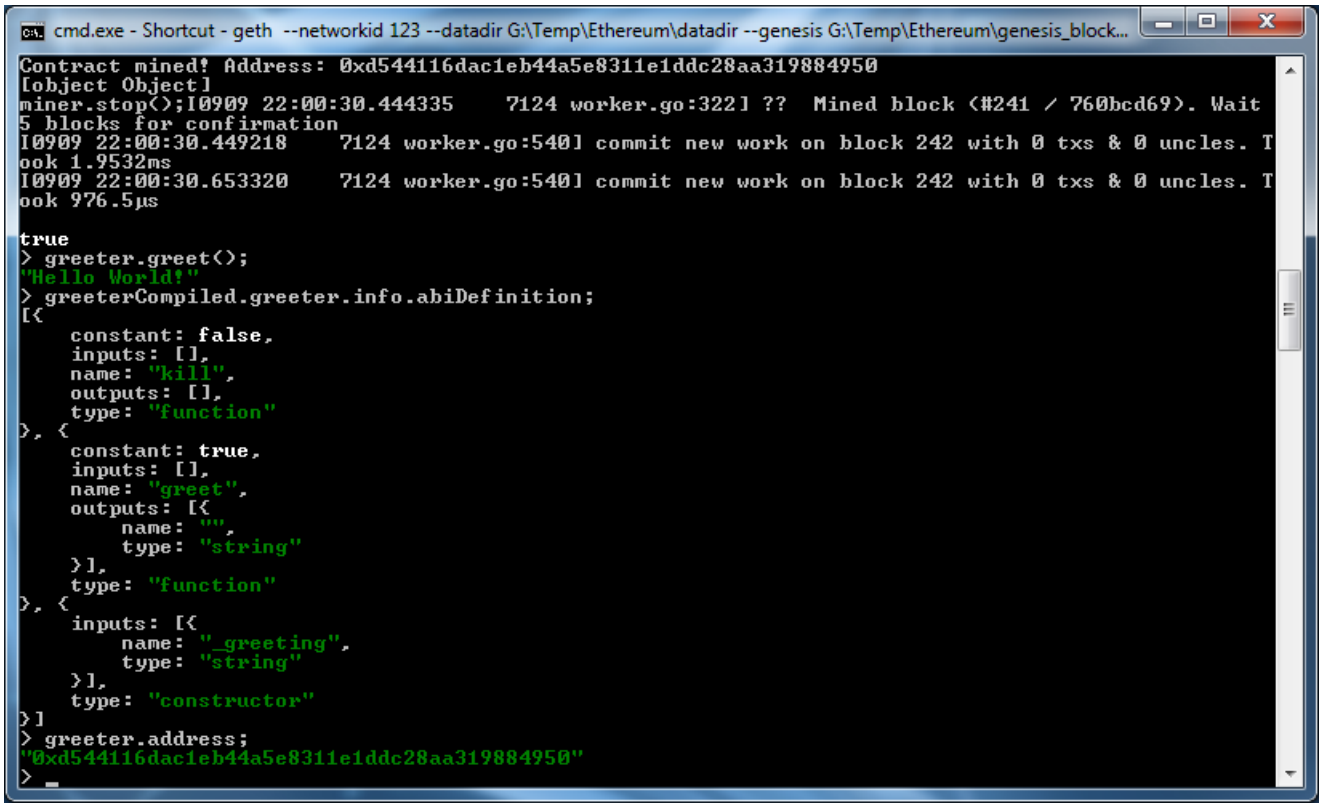
Procedure

- Create Contract
- (Execute below commands)

Get ABI and Address

```
var greeterABI     = greeterCompiled.greeter.info.abiDefinition;
var greeterAddress = greeter.address;
var greeter2       = eth.contract(greeterABI).at(greeterAddress);
greeter2.greet();
```

Command Prompt



greeterABI

```
[{
    constant: false,
    inputs: [],
    name: "kill",
    outputs: [],
    type: "function"
}, {
    constant: true,
    inputs: [],
    name: "greet",
    outputs: [{
        name: "",
        type: "string"
    }],
    type: "function"
}, {
    inputs: [{
        name: "_greeting",
        type: "string"
    }],
    type: "constructor"
}]
```

greeterAddress

```
"0xd544116dac1eb44a5e8311e1ddc28aa319884950"
```

2.1.6.5 Kill

Info [R]

- This tutorial shows how to kill Contract by calling its kill function which has to be implemented by each Contract.
- Calling kill function creates transaction which needs to be mind in order to get executed.
- Kill can only be triggered by a transaction sent from the contracts owner which is given as the only transaction parameter.

Procedure

- Create Contract
- (Execute below commands)

Kill Contract

```
greeter.kill.sendTransaction({from:eth.accounts[4]});
```

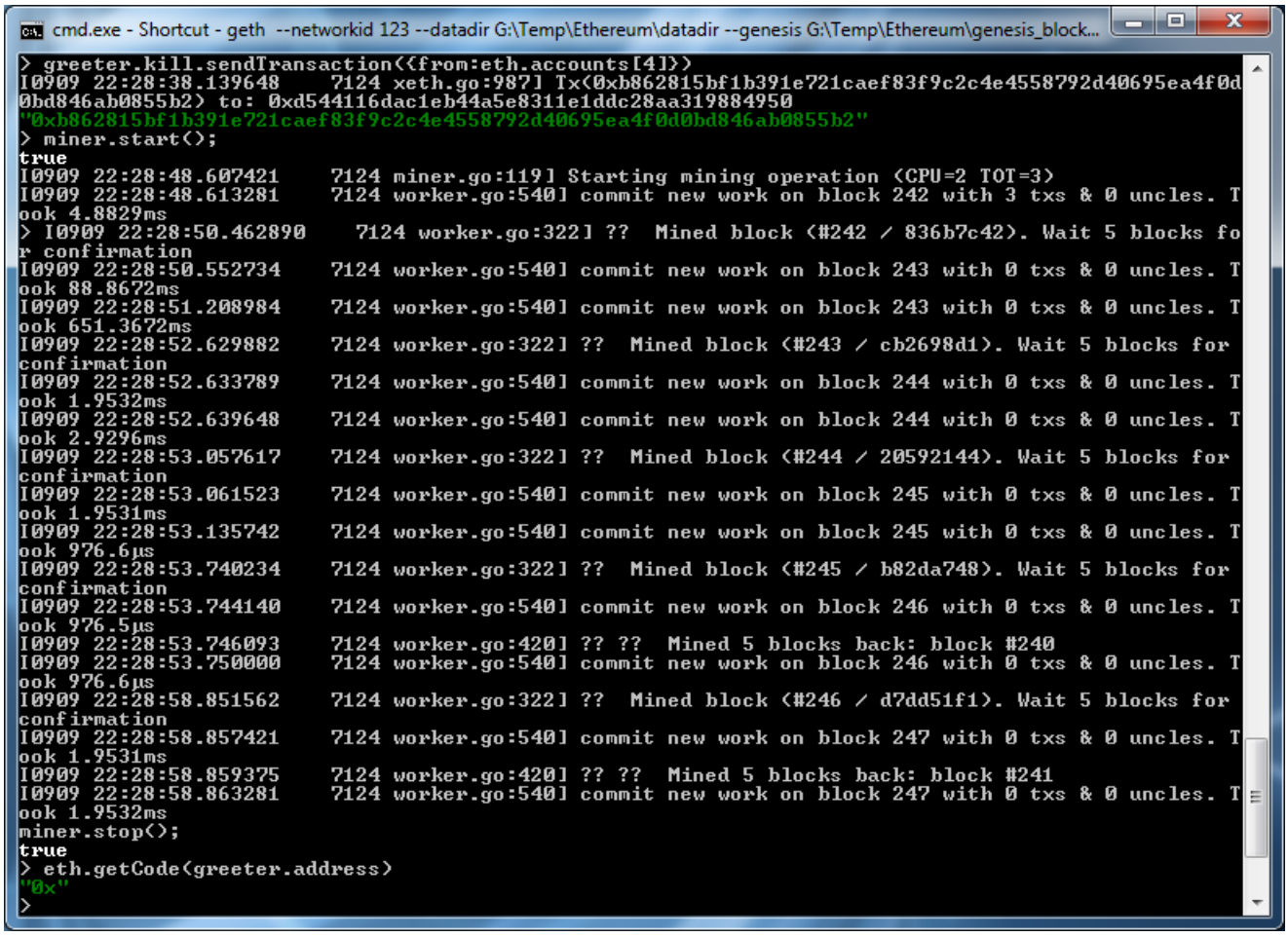
Start mining. Wait a while. Stop mining.

```
miner.start();
miner.stop();
```

Check if Contract was killed. It should return "0x".

```
eth.getCode(greeter.address);
```

Command Prompt



2.2 ETH

Info

- ETH is graphical user interface for running a full Ethereum node implemented in C++.

2.2.1 Setup

Info

- Following tutorials show how to install and setup GETH Client.

2.2.1.1 Install

Info [R]

- This tutorial shows how to Install ETH.

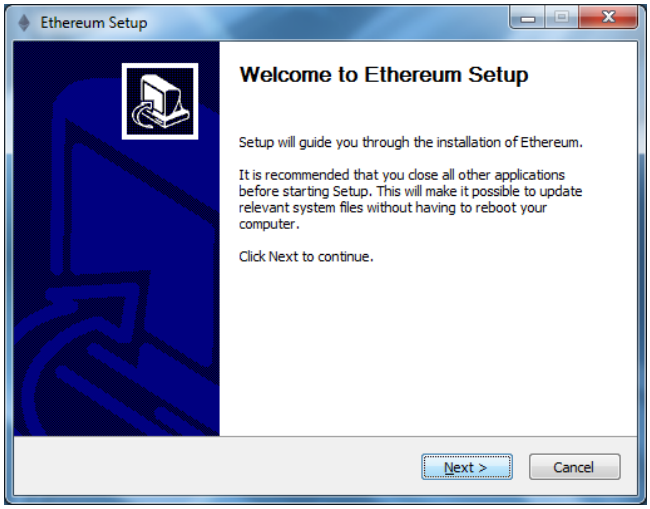
Download

- https://build.ethdev.com/builds/Windows%20C%2B%2B%20master%20branch/Ethereum-win64-latest.exe
- Save as: G:\Downloads\Ethereum-win64-latest.exe

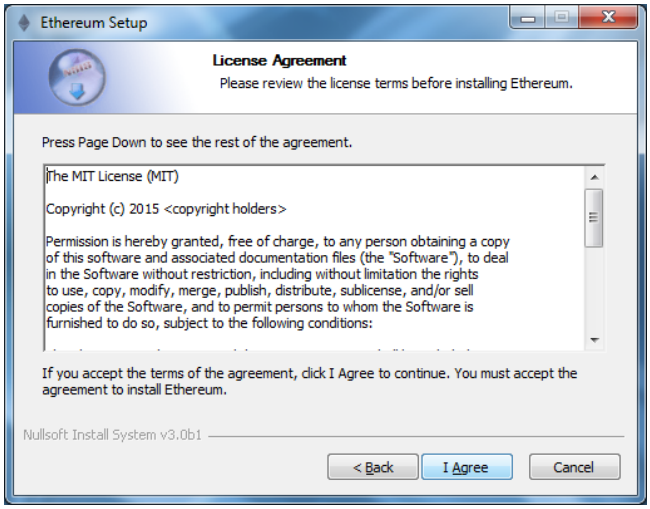
Install

- Execute G:\Downloads\Ethereum-win64-latest.exe
- Next
- I Agree
- Add Ethereum to the system PATH to all users: SELECT – Next
- Destination Folder: I:\Installed\Programming\Ethereum 0.9.41 – Next
- Next
- Install
- Finish

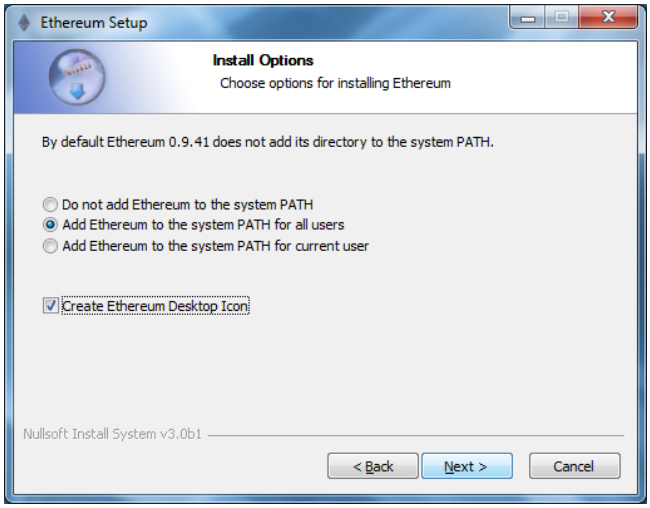
Next



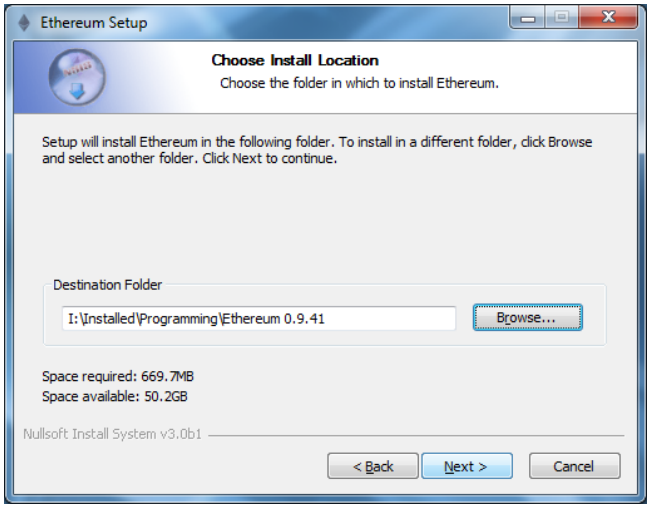
I Agree



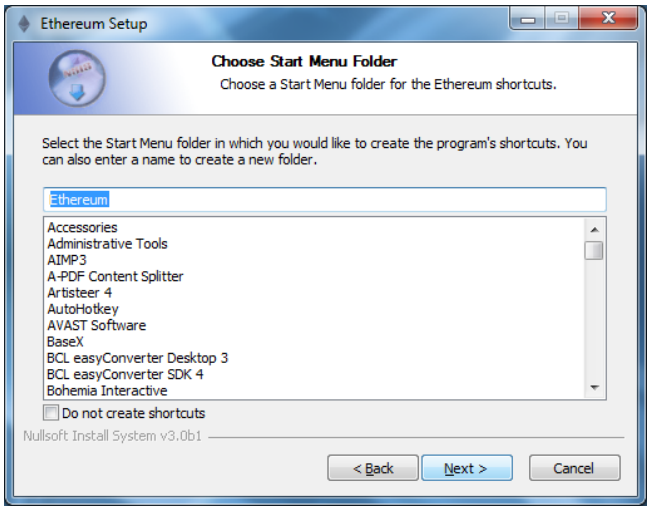
Next



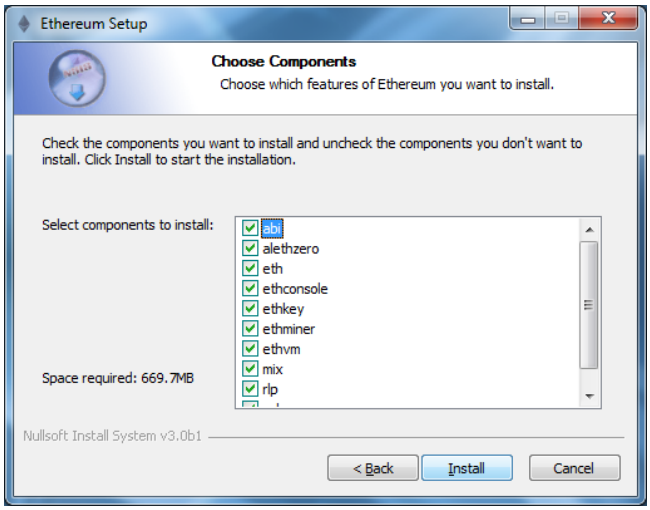
Next



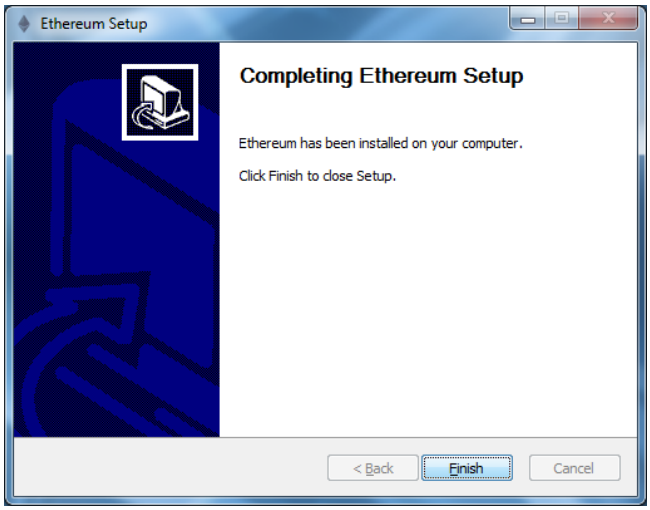
Next



Install



Finish



2.2.1.2 Edit PATH Environment Variable

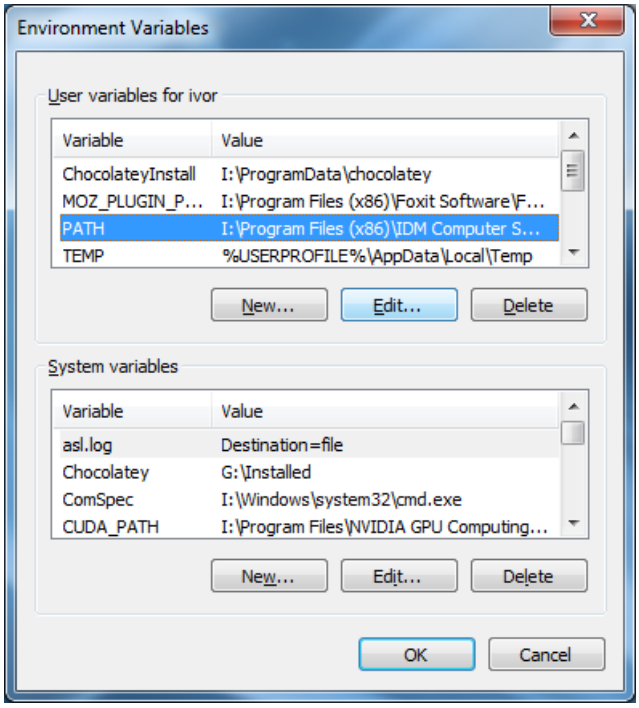
Info

- This tutorial shows how to edit PATH Environment Variable by appending directory where eth.exe is installed.
- This will allow you to execute eth from any directory.
- Without this step you would first need to position yourself in directory where ETH is installed by calling `I:\Installed\Programming\Ethereum 0.9.41\Release` before being able to run ETH inside Command Prompt.

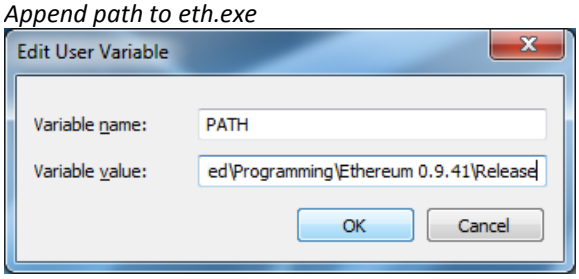
Procedure

- Show Environment Variables
- User Variables
- PATH
- Edit
- Variable value: **(Append ;I:\Installed\Programming\Ethereum 0.9.41\Release)**
- 3*OK

Edit PATH



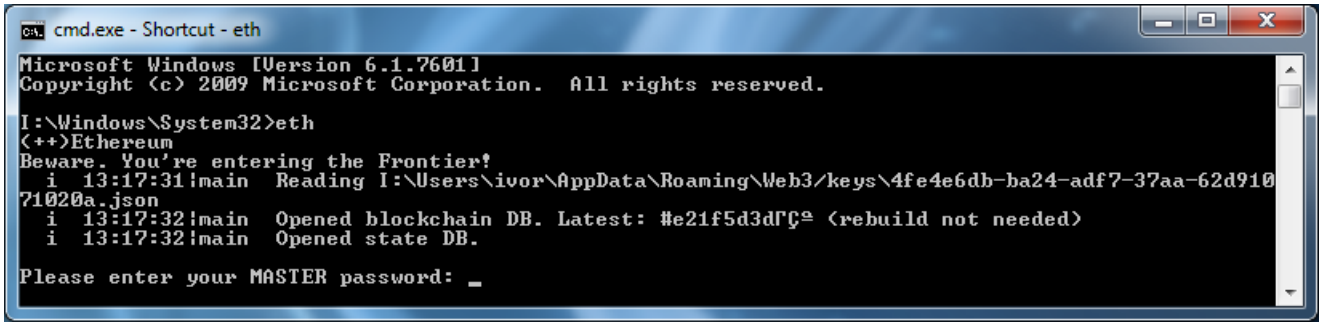
Append path to eth.exe



Test

- Start Command Prompt
- `eth`

Command Prompt



3 Solidity

Info

- Solidity is high level programming language especially designed for implementing Ethereum contracts.
- Following tutorials explain Solidity syntax in detail.
- Syntax of a programming language is the set of rules that define the combinations of symbols that are considered to be correctly structured programs in that language.

Variable

- Variable is a named memory location reserved to store a number in it and has following parameters
  - Identifier is the name of the variable, the name of memory locaton
  - Address is the beginning of that memory location inside computer memory.
  - Data Value is the number (zeros and ones) stored at that memory location, also known as just Data or Value.
  - Data Type is type of data that is stored in that memory location defining what these zeros and ones represent

Data Type

- Data type tells computer how to intepret number stored in memory location.
- This is needed because number stored in a memory location is just zeros and ones and the same combination of zeros and ones can have different meaning depending on what they are supposed to represent.
- For instance the same combination of zeros and ones can represent either integer 68656c6c6f or string "Hello".
- So in order for print(myvariable) function to properly display data stored in memory location it needs to know if data represents integer or string so that it can display either 68656c6c6f or "Hello" respectivly on the screen.
- Data type also defines the size of memory location so that computer would know how many zeros and ones to take beginning from the address of the memory location.

Example

- In the following example
  - Identifier is `name`
  - Data Value is `"Ivor Online"` and it is defined with String literal
  - Data Type of data that is actually being stored is defined by String literal `"Ivor Online"` and it is of data type String
  - Address is assigned automatically by computer and is unknown to us
  - = is assignment operator used to store value into variable
- With string literal `"Ivor Online"` we have defined data value, zeroes and ones that should go into memory location, but we have also defined that these zeroes and ones are of String data type.

Store data into variable

```
string name = 'IvorOnline'
```

Literal

- **Literal is string representation of data type and value like**
  - "Ivor Online" is string literal since it represents string data type of value Ivor Online
  - 'Ivor Online' is also string literal, but using single quotes notation, since it also represents string data type
  - 10 is integer literal since it represents integer data type of value 10
  - 0x0A is also integer literal, but using different notation, since it also represents integer data type of value 10
- Literal notations are different ways of constructing literals that represent the same data type.
  For example following integer literals represent the same data type (integer) using different literal notations
  10, +10, 012 or 0xA.

3.1 Comments

Info **[R]**

- Comments are pieces of text inserted into source code to clarify it.
- They are not instructions for the computer and are therefore ignored by execution engine.
- Specially formatted comments can be used to generate documentation directly from the source code.
- Solidity supports two types of comments as shown in below table.

Comment types

TYPE	DESCRIPTION
Single Line	`//Single line comment`
Multi Line	`/* Multi line comment.` `Second line. */`

3.1.1 Single Line

Info [R]

- Single Line comments can't span over multiple lines.
- Start of single line comment is indicated by especially reserved combination of characters `//`.
- End of single line comment is indicated by the end of the line.
- Benefit of single line comments is that they do not need to be terminated with terminator keyword.
- Downside of single line comment is that creating longer comments requires placing indicator for start of comment at the beginning of each line.

Syntax

COMMENT SYNTAX	EXAMPLE
//...	`//Single line comment`

Example

```
contract greeter is mortal {

  function greet() returns (string) {
    string username = 'IvorOnline'; //Account username
    string password = 'mypassword'; //Account password
    return username;
  }

}
```

3.1.2 Multi Line

Info [R]

- Multi Line comments can span over multiple lines.
- Start of multi-line comment is indicated by combination of slash and multiply character `/*`.
- End of multi-line comment is indicated by combination of multiply and slash character `*/`.
- Benefit of multi-line comments is that they allow you to write a lot of text that spans over multiple lines without having to prefix each line with predefined prefixes as is the case with single line comments.
- Downside of multi-line comments is that they have to be terminated with specific indicator which presents typing overhead compared to single line comments for shorter text.

Syntax

COMMENT SYNTAX	EXAMPLE
/* ... */	/* Multi line comment. Second line. */

Example

```
contract greeter is mortal {

  function greet() returns (string) {
    /* User account is defined by
       username and pasword       */
    string username = 'IvorOnline';
    string password = 'mypassword';
    return username;
  }

}
```

3.2 Operators

Info

- Operators are used to assign, combine or compare values of variables.

3.2.1 Assignment

Info [R]

- Assignment operators assign value to variable.
- Shortcut assignment operators, like += and /+, perform some mathematical operation before making the assignment.

Logical operators

OPERATOR	EXAMPLE	NAME	DESCRIPTION
=	a = 5	Assignment	a = 5
+=	a += 5	Addition-Assignment	a = a + 5
-=	a -= 5	Subtraction-Assignment	a = a - 5
*=	a *= 5	Multiplication-Assignment	a = a * 5
/=	a /= 5	Division-Assignment	a = a / 5
%=	a %= 5	Modulus-Assignment	a = a % 5

Example

```
contract greeter is mortal {

  function test() returns (int) {

    //Test variables.-----------------------------------------------------------------------------------
    int x = 15;

    //Assignement operators.----------------------------------------------------------------------------
    x += 10;  // x = x + 10;
    x -= 10;  // x = x - 10;
    x *= 10;  // x = x * 10;
    x /= 10;  // x = x / 10;
    x %= 10;  // x = x % 10;

    //Display result.-----------------------------------------------------------------------------------
    return x; // Replace add with any other variable from above to see its value.

  }

}
```

3.2.2 Comparison

Info [R]

- Comparison operators are used to compare values of two variables.
- Result is always a boolean data type indicating if comparison was true or false.
- Comparison operators provide a method to direct program flow depending on the outcome.

Comparison operators

OPERATOR	EXAMPLE	NAME	DESCRIPTION
==	a == b	Equal	TRUE if a is equal to b.
!=	a != b	Not equal	TRUE if a is not equal to b.
<	a < b	Less then	TRUE if a is strictly less than b.
>	a > b	Greater then	TRUE if a is strictly greater than b.
<=	a <= b	Less then or equal	TRUE if a is less than or equal to b.
>=	a >= b	Greater then or equal	TRUE if a is greater than or equal to b.

Example

```
contract greeter is mortal {

  function test() returns (bool) {

    //Test variables.----------------------------------------------------------------------------------
    int left  = 65;
    int right = 70;

    //Comparison operations.---------------------------------------------------------------------------
    if (left == right) { return true; } // Left    equals              right.
    if (left != right) { return true; } // Left is different from      right.

    if (left <  right) { return true; } // Left is smaller   then      right.
    if (left >  right) { return true; } // Left is greater   then      right.
    if (left <= right) { return true; } // Left is smaller   or equals right.
    if (left >= right) { return true; } // Left is greater   or equals right.

  }

}
```

3.2.3 Arithmetic

Info [R]

- Arithmetic operators are used to perform mathematical operations.
- Mathematical operations, not supported by arithmetic operators, can be performed using PHP mathematical functions.
- Division ("/") returns integer if both operands are evenly divisible integers (or strings that get converted to integers).
- In all other cases float is returned.
- Operands of modulus ("%") are converted to integers (by stripping the decimal part) before processing.
- Result of the modulus ("%") has the same sign as the dividend (first parameter $a).

Arithmetic operators

OPERATOR	EXAMPLE	NAME	DESCRIPTION
unary -	-a	Negation	Changes sign of a.
unary +	+a		Does no change to the value of a.
+	a + b	Addition	Adds a and b.
-	a - b	Subtraction	Subtracts b from a.
*	a * b	Multiplication	Multiply a with b.
/	a / b	Division	Divide a by b.
%	a % b	remainder	Return reminder of dividing a by b.
**	a ** b	Exponentiation	Returns a to the exponent of b.
++	a++	Post-increment	Increment a by one after it is used in expression.
--	a--	Post-decrement	Decrease a by one after it is used in expression.
++	++b	Pre-increment	Increment b by one before it is used in expression.
--	--b	Pre-decrement	Decrease b by one before it is used in expression.

Example

```
contract greeter is mortal {

  function test() returns (int) {

    //Test variables.----------------------------------------------------------------------------------
    int x          = 10;
    int y          = 20;

    //Arithmetic operations.---------------------------------------------------------------------------
    int negate     =    -y;    //  -20 = -20
    int add        = x + y;    //10+20 =  30
    int subtract   = x - y;    //10-20 = -10
    int multiply   = x * y;    //10*20 = 200
    int divide     = x / y;    //10/20 = 0.5
    int modulo     = x % y;    //10%20 = 0*20+10 = 10
    int increment2 =     y++;  //Store y into increment2 and then increment y by 1.
    int decrement2 =     y--;  //Store y into decrement2 and then decrement y by 1.
    int increment1 =   ++y;    //Increment y by 1 and then store y into increment1.
    int decrement1 =   --y;    //Decrement y by 1 and then store y into decrement1.

    //Return result.-----------------------------------------------------------------------------------
    return(increment2);        //Replace add with any other variable from above to see its value.

  }

}
```

3.2.4 Bitwise

Info [R]

- Bitwise operators combine bits within one or two integers.
- Right shift `>>n` is Arithmetic Shift equivalent of dividing by 2^n which preserves operand sign.
- AND, OR, XOR and NOT do not change the value of operands.

Bitwise operators

OPERATOR	EXAMPLE	NAME	DESCRIPTION
&	a & b	AND	Set bits that are set in both a and b
|	a | b	OR	Set bits that are set in either a or b.
^	a ^ b	XOR	Set bits that are different in a and b. Exclusive OR.
~	~ b	NOT	Invert bits in b. Negation.
<<	$a << $b	Shift left	Shift the bits of a by b steps to the left (each step means "multiply by two")
>>	$a >> $b	Shift right	Shift the bits of a by b steps to the right (each step means "divide by two")

Example

```
contract greeter is mortal {

  function test() returns (byte) {

  //Test variables.----------------------------------------------------------------
  byte left       = 27;             //11011
  byte right      = 18;             //10010

  //Bitwise operators.-------------------------------------------------------------
  byte and        =  left & right;  //10010.    1 if both   bits are 1.
  byte or         =  left | right;  //11011.    1 if either bit  is  1.
  byte xor        =  left ^ right;  //01001.    1 if        bits are different.
  byte invert     = ~left;          //11111111111111111111111111100100.  Invert bits.
  byte shiftLeft  =  left << 3;     //11011000. Shift bits to left  by 3 positions. Fill with 0.
  byte shiftRight =  left >> 2;     //110.      Shift bits to right by 2 positions. Fill with sign bit.

  //Display results.---------------------------------------------------------------
  return shiftRight;

  }

}
```

3.2.5 Logical

Info [R]

- Logical operators are used to compare variable values and the result is always bool.

Logical operators

OPERATOR	EXAMPLE	NAME	DESCRIPTION
&&	a && b	AND	True if both a and b are true.
||	a || b	OR	True if either a or b is true.
!	!a	NOT	True if b is not true.

Example

```
contract greeter is mortal {

  function test() returns (bool) {

    //Test variables.----------------------------------------------------------------------------------------
    int left  = 65;
    int right = 70;

    //Logical operators.-------------------------------------------------------------------------------------
    if (          !   (left==80)) { return true; } //left is    NOT equal to 80
    if ( left> 50 &&  right==70 ) { return true; } //left >  50 AND right == 70
    if ( left!=50 ||  right> 90 ) { return true; } //left != 50 OR  right >  90

  }

}
```

3.3 Statements

Info

- Statement is smallest part of code which can be executed.
- Statements are classified depending on their function.

Conditional Statements

- Conditional Statements define which part of code should be executed depending on some condition.
- Parentheses can not be omitted for condition.
- Parentheses can be omitted for the code only if code to be executed contains single statement.
- Conditional Statements are further classified as
  - Conditional Branching where if condition is true code is executed only ones.
  - Conditional Looping statements where as long as the condition is true code is executed through multiple iterations.

Syntax

```
keyword ( condition ) { code }
```

3.3.1 Conditional Branching

Info

- Conditional branching statements are used to control program flow based on specified condition being true or false.
- If condition is true code is executed only ones.

Syntax

```
keyword ( condition ) { code }
```

3.3.1.1 if

Info

- `if` statement executes specified code if specified condition is true.
- If code contains only single statement, curly brackets can be omitted.
  This is not recommended since it makes code less readable and adding additional code statements can lead to errors.

Syntax

```
if (condition) { code }
```

Example

```
contract greeter is mortal {

  function test() returns (bool) {

    int number = 10;

    if (number == 10) { return true; }

  }

}
```

3.3.1.2 else

Info

- `else` statement can be used only in combination with `if` statement.
- If condition specified by `if` statement is TRUE then only code defined in `if` statement is executed.
- If condition specified by `if` statement is FALSE then only code defined in `else` statement is executed.
- If code belonging to `if` or `else` statements contains only single statement, curly brackets can be omitted.
  This is not recommended since it makes code less readable and adding additional code statements can lead to errors.

Syntax

```
if   (condition) { code1 }
else             { code2 }
```

Example

```
contract greeter is mortal {

  function test() returns (bool) {

    int number = 10;

    if   (number == 10) { return true;  }
    else                { return false; }

  }

}
```

3.3.1.3 else if

Info

- Multiple `else if` statements can be used in combination with `if` statement.
- `if...else if` combo works as multiple `if` statements.
- Conditions are evaluated in sequence and only code belonging to the first condition which evaluates to true is executed.
- `if...else if` combo can optionally end with `else` statement.
  If so, then if all conditions evaluate to FALSE only code belonging to `else` statement will be executed.

Syntax

```
if      (condition1) { code1  }
else if (condition2) { code2  }
...
else                 { code10 }
```

Example

```
contract greeter is mortal {

  function test() returns (bool) {

    int number = 10;

    if      (number == 10) { return true;  }
    else if (number == 11) { return true;  }
    else                   { return false; }

  }

}
```

3.3.2 Conditional Looping

Info

- Looping statements are used to control program flow based on specified condition being true or false.
- As long as the condition is true code is being executed through multiple iterations.

3.3.2.1 for

Info

- Depending on the condition, `for` statement can execute specified code multiple times.
  - `expression1` is executed at the beginning of the first iteration.
  - `condition` is evaluated at the beginning of each iteration and
    - if it evaluates to TRUE code is executed.
    - if it evaluates to FALSE code is NOT executed and program breaks out of `for` loop.
  - `expression2` is executed at the end of each iteration.
- Both `expressions` and `condition` can be left empty in which case `condition` evaluates to TRUE.
- You can break out of such `for` loop by using conditional and `break` statement inside a code.

Syntax

```
for (expression1; condition; expression2) { code }
```

Example

```
contract greeter is mortal {

  function test() returns (int) {

    //SIMPLE.   Output is: 4.-------------------------------------------------------------------------------
    int cnt1;
    for (int i=1; i<=4; i++) {
      cnt1++;
    }

    //BREAK.     Output is: 2  since we break out of for loop when i=3.-----------------------------------
    int cnt2;
    for (int j=1; j<=4; j++) {
      if(j==3) { break; }
      cnt2++;
    }

    //CONTINUE. Output is: 3 since k = 3 is skipped.-------------------------------------------------------
    int cnt3;
    for (int k=1; k<=4; k++) {
      if(k==3) { continue; }
      cnt3++;
    }

    //EMPTY.    Output is: 4 since we break on 5.----------------------------------------------------------
    int cnt4;
    int l = 1;
    for (;;) {
      if (l==5) { break; }
      cnt4++;
      l++;
    }

    return cnt4;

  }

}
```

3.3.2.2 while

Info

- `while` statement iteratively executes specified code as long as specified condition is TRUE.
- `condition` is evaluated at the beginning of each iteration and
  - If it evaluates to TRUE code is executed.
  - If it evaluates to FALSE code is NOT executed and program breaks out of `while` loop.

Syntax

```
while (condition) { code }
```

Example

```
contract greeter is mortal {

  function test() returns (int) {

    //SIMPLE. Output:4.
    int i = 1;
    while (i<=4) {
      i++;
    }

    //BREAK.  Output:3 since we break out of for loop when j=3.
    int j = 1;
    while  (j<=4){
      j++;
      if (j==3) { break; }
    }

    //NEXT. Output:5 since step 3 is skipped.
    int k = 0;
    while (k<=4){
      if (k==3) { continue; }
      k++;
    }

    return k;

  }

}
```

3.3.3 Unconditional Jumping

Info

- Jumping statements are used to unconditionally continue code execution at some other part of code.
- Use of such statements is considered bad practice since they make it harder to follow code execution.
- You should use conditional statements instead.

3.3.3.1 break

Info

- `break` statement is used looping statements to end execution of for and while loops.

Syntax

```
break;
```

Example

```
contract greeter is mortal {

  function test() returns (int) {

    //Break from for loop.
    int cnt;
    for (int i=1; i<=4; i++) {
      if (j==3) { break; }
      cnt++;
    }

    //Break from while loop.
    int j = 1;
    while  (j<=4){
      j++;
      if (j==3) { break; }
    }

    return cnt;

  }

}
```

3.3.3.2 continue

Info

- `continue` statement is used inside looping statements to stop current iteration and continue with the next one.
- This means that you can use it to continue with the next iteration in for and while loops.

Syntax

```
continue;
```

Example

```
contract greeter is mortal {

  function test() returns (int) {

    //Continue with the next iteration.-------------------------------------------------------------------------
    int cnt;
    for (int i=1; i<=4; i++) {
      if(i==3) { continue; }
      cnt++;
    }

    //Continue with the next iteration.-------------------------------------------------------------------------
    int k = 0;
    while (k<=4){
      if (k==3) { continue; }
      k++;
    }

    return cnt;

  }

}
```

3.4 Functions

Info

- Function is block of code that can be repeatedly called with different parameters and which can return some value.
- In Solidity functions can only be declared inside a contract, there are no global functions.
- Therefore we can also refer to Solidity functions as methods since contracts are equivalent to classes and functions declared inside classes are called methods.

Function declaration

- Function is declared
  - inside a contract
  - using keyword `function`
  - followed by the function name
  - followed by the list of input parameters inside curved brackets with mandatory types and optional names
  - followed by optional keyword `returns` and type of output value inside curved brackets (if function returns value)
  - followed by the function body inside curly brackets

Syntax

```
function name(dataType paramName, ...) returns (dataType) { body }
```

Example

```
contract greeter is mortal {

  //Function declaration.
  function calculate(int x, int y) private returns (int) {
    return x + y;
  }

  //Function call.
  function greet() returns (int) {
    return calculate(10, 5);
  }

}
```

3.4.1 Name

Info

- This tutorial shows how to declare function name.
- Declaration of function name should follow keyword `function`.
- Function name is used to uniquely identify function.
- You can't have two functions with the same name even if they have different number or type of parameters.

Example

```
contract greeter is mortal {

  function test() returns (int) {
    int age = 50;
  }

}
```

3.4.2 Return values

Info

- This tutorial shows how to declare function return a value.
- Return value is declared by using `return` keyword which can be used multiple times on different places inside function.
- If you want to return more than one value you can store them in an array or an object.
- You have to define data type returned by function.

3.4.2.1 None

Info

- This tutorial shows how to create and call method that has no return value.
- To declare function which returns no value you should simply omit using `return` keyword inside a function.

Example

```
contract greeter is mortal {

  function test() returns (int) {
    int age = 50;
  }

}
```

3.4.2.2 Single

Info

- This tutorial shows how to return single value from a function.
- Data type of returned value must be defined using keyword `returns` followed by data type specified inside brackets.
- Return value is declared using `return` statement which can be used multiple times on different places inside function.

Example using single return statement

```
contract greeter is mortal {

  function getAge() returns (int) {
    int age = 50;
    return age;
  }

  function greet() returns (int) {
    return getAge();
  }

}
```

Example using multiple return statements

```
contract greeter is mortal {

  function getAge() returns (int) {

    int number = 10;

    if      (number == 10) { return 10;  }
    else if (number == 11) { return 11;  }
    else                   { return 100; }

  }

  function greet() returns (int) {
    return getAge();
  }

}
```

3.4.2.3 Multiple

Info

- This tutorial shows how to return multiple values from a method.
- Method can only return single value so if you want to return multiple values you can store them in an
  - array which allows you to return multiple values of the same types
  - struct which allows you to return multiple values of different types

Example using struct

```
contract greeter is mortal {

  struct Account {
    bool acess;
    int  age;
  }

  function getAccount() private returns (Account) {

    Account ivor;
    ivor.acess = true;
    ivor.age   = 50;

    return ivor;

  }

  function greet() returns (int) {
    return getAccount().age;
  }

}
```

Example using array

```
contract greeter is mortal {

  function getAccount() private returns (int[2]) {

    int[2] ivor;
    ivor[0] = 1;
    ivor[1] = 50;

    return ivor;

  }

  function greet() returns (int) {
    return getAccount()[0];
  }

}
```

3.4.3 Parameters

Info

- Function can have zero or more parameters.
- It can also have variable number of parameters meaning that not all parameters have to be given when calling function.
- Parameters which were not defined during function call will be given default values.
- For each parameter data type must be specified.

3.4.3.1 None

Info

- This tutorial show how to declare function that has no input parameters.
- If method should have no parameters simply leave the area between curved braces following function name empty.

Example

```
contract greeter is mortal {

  //Declare method.
  function test() private returns (int) {
    return 50;
  }

  //Call method.
  function greet() returns (int) {
    return test();
  }

}
```

3.4.3.2 Multiple

Info

- This tutorial shows how to declare function that has fixed number of input parameters.
- For each parameter you have to define type and optional name.
- If parameter was not given a name you won't be able to use it inside the function.
  And you also won't be able to call the function using JSON object.
- When such function is called you have to either provide
  - list of parameters with the exact number of parameters in the order in which they are declared
  - JSON object with parameter name-value pairs given in any order

Example

```
contract greeter is mortal {

  //Multple parameters.
  function calculate(int x, int y) private returns (int) {
    return x + y;
  }

  //List of parameters.
  function greet() returns (int) {
    return calculate(10, 5);
  }

  //JSON object with parameter name-value pairs.
  function greet2() returns (int) {
    return calculate( {x:2, y:3} );
  }

}
```

Example with unnamed parameter

```
contract greeter is mortal {

  //Multple parameters.
  function calculate(int x, int) private returns (int) {
    return x;
  }

  //List of parameters.
  function greet() returns (int) {
    return calculate(10, 5);
  }

}
```

3.4.4 Type

Info

- Function type can define from where the function can be referenced by using keywords
  - `public` which defines that function can be called outside the contract (default scope)
  - `private` which defines that function can be called only from other functions inside the same contract
- Function type can also define if function can change state of the contract by using keyword
  - `constant` which defines that function is not allowed to modify the state of the contract

3.4.4.1 public

Info

- This tutorial show how to declare a public function which can be called outside the contract (default scope).
- Public functions represent contract's interface to the outside world, exposed functions that can be called.

Example

```
contract greeter is mortal {

  //Declare public method accessed from outside the contracts.
  function greet() public returns (int) {
    return 50;
  }

}
```

3.4.4.2 private

Info

- This tutorial show how to declare private function which can be called only from other functions inside the same contract
- Private functions represent contract's implementation and are invisible to the outside world.

Example

```
contract greeter is mortal {

  //Declare method accessed from inside the contract.
  function test() private returns (int) {
    return 50;
  }

  //Declare public method accessed from outside the contracts.
  function greet() public returns (int) {
    return test();
  }

}
```

3.4.4.3 constant

Info

- This tutorial show how to declare constant function which is not allowed to modify the state of the contract.

Example

```
contract greeter is mortal {

  //Declare constant method which is not allowed to modify the state of the contract.
  function greet() constant returns (int) {
    return 50;
  }

}
```

3.4.5 Reference

Info

- Function can be referenced by using
  - Internal Function Call from the function in the same contract which is done by simply using its name
  - External Function Call from the function in another contract which is done by referencing contract that contains function that is being called by using contract address

3.4.5.1 Internal Function Call

Info

- This tutorial show how to reference function with Internal Function Call, from another function in the same contract.
- This is done simply by using function name followed by the list of parameters.

Example

```
contract greeter is mortal {

  //Multple parameters.
  function calculate(int x, int y) private returns (int) {
    return x + y;
  }

  //List of parameters.
  function greet() returns (int) {
    return calculate(10, 5);
  }

}
```

3.5 Contracts

Info

- Contracts are equivalent to classes in other OOPL (Object Oriented Programming Languages).
- They both have constructor, member variables, methods and can inherit from each other.
- Solidity contracts are deployed to the block chain as their permanent storage and are referenced by their address.

Contract declaration

- Contract is declared
  - using keyword `contract`
  - followed by the contract name
  - followed by optional keyword `is` and the name of contract it inherits from (if inheritance is used)
  - followed by the contract body inside curly brackets

Syntax

```
contract name is inheritedContractName { body }
```

Example

```
contract greeter is mortal {

  function greet() returns (int) {
    return 10;
  }

}
```

3.5.1 Name

Info

- This tutorial shows how to declare contract's name.
- Declaration of contract's name should follow keyword `contract`.

Example

```
contract greeter is mortal {

  function test() returns (int) {
    int age = 50;
  }

}
```

3.5.2 Fields

Info

- Fields are variables defined inside the contract but outside any method.
- Fields can have scope modifiers defining from where the field can be referenced where
  - `public` keyword allows field to be accessed from outside the contract
  - `private` keyword allows field to be accessed only from the methods of the same contract

Example

```
contract greeter is mortal {

  //Field.
  int private age = 50;

  //Method.
  function greet() returns (int) {
    return age;
  }

}
```

3.5.2.1 public

Info

- This tutorial show how to declare a public field which can be referenced outside the contract (default scope).
- Fields are public by default.

Example

```
contract greeter is mortal {

  //Field.
  int public age = 50;

}
```

output

```
Returned: "50"
```

3.5.2.2 private

Info

- This tutorial show how to declare private field which can be referenced only from functions & fields of the same contract.

Example

```
contract greeter is mortal {

  //Field.
  int private age = 50;

  //Method.
  function greet() returns (int) {
    return age;
  }

}
```

3.5.3 Methods

Info

- In Solidity all functions must be defined inside a contracts which means that all functions are methods.
- Therefore everything that was said for Functions can be applied to methods.

Example

```
contract greeter is mortal {

  //Method.
  function greet() returns (int) {
    return 10;
  }

}
```

3.5.4 Constructor

Info

- Constructor is special kind of method that has the same name as the contract.
- It is invoked only once when instantiating the contract after which it can't be called again on that instance.
- It is used to initialize created instance for example by setting field values as shown in below example.

Example

```
contract greeter is mortal {

  //Field.
  int private age;

  //Constructor.
  function greeter (int ageParam) {
    age = ageParam;
  }

  //Methos.
  function test() returns (int) {
    return age;
  }

}
```

3.5.5 Inheritance

Info

- Contracts support multiple inheritance allowing contracts to inheriting fields and methods of the parent contracts.

Example

```
contract mortal {
  address owner;                                                  /* Define address variable owner    */
  function mortal() { owner = msg.sender; }                       /* Set owner during initialization  */
  function kill()   { if (msg.sender == owner) suicide(owner); }  /* Get funds on killing contract    */
}

contract singer {
  function sing() returns (int) { return 10; }
}

contract greeter is mortal, singer {
  string greeting;                                                 /* Define string variable greeting */
  function greeter(string _greeting) public  { greeting = _greeting; }  /* Run during initialization       */
  function greet() constant returns (string) { return greeting;      }  /* Main function                   */
}
```

3.5.1 Data Types

Info **[R]**

- Solidity is a statically typed language, which means that the type of each variable needs to be specified at compile-time.
- Solidity provides several elementary types which can be combined to complex types like arrays.
- Variables of value types are always passed by value, they are always copied when they are used as function arguments

bool, int, uint, address, byte

3.5.2 Scalar

Info

- Scalar datatypes are capable of containing a single item of information.
- Scalar datatypes include: bool, int, uint, address, byte

3.5.2.1 bool

Info **[R]**

- Bool variable can have values true or false.
- Bool variables can be combined using logical operators shown in the table below.
- They support Logical Operators.

Boolean

INTEGER SIZE	DESCRIPTION
bool	Can have values **true** or **false**

Example

```
contract greeter is mortal {

  function greet() returns (bool) {
    bool   accessGranted = true;
    return accessGranted;
  }

}
```

output

```
Returned: true
```

3.5.2.2 int

Info [R]

- Int variable can store signed integers of various sizes as shown in the table below.
- Int is alias for int256.

Singed integers

INTEGER SIZE	DESCRIPTION
int8	Signed integer with 8 bits
int16	Signed integer with 16 bits
int32	Signed integer with 32 bits
int64	Signed integer with 64 bits
int128	Signed integer with 128 bits
int256 or int	Signed integer with 256 bits

Example

```
contract greeter is mortal {

  function greet() returns (int) {
    int    temperature = -50;
    return temperature;
  }

}
```

output

```
Returned: "-50"
```

3.5.2.3 uint

Info [R]

- Uint variable can store signed integers of various sizes as shown in the table below.
- Uint is alias for uint256.

Unsinged integers

INTEGER SIZE	DESCRIPTION
uint8	Unsigned integer with 8 bits
uint16	Unsigned integer with 16 bits
uint32	Unsigned integer with 32 bits
uint64	Unsigned integer with 64 bits
uint128	Unsigned integer with 128 bits
uint256 or uint	Unsigned integer with 256 bits

Example

```
contract greeter is mortal {

  function greet() returns (uint) {
    uint   age = 50;
    return age;
  }

}
```

output

```
Returned: "50"
```

3.5.2.4 address

Info [R]

- Address variable holds 20 byte value that represents an Ethereum address.
- They support Comparison Operators.

Address

INTEGER SIZE	DESCRIPTION
address	Holds 20 byte value that represents an Ethereum address

Example

```
contract greeter is mortal {

  function greet() returns (address) {
    address sender = msg.sender;
    return  sender;
  }

}
```

output

```
Returned: "0xca35b7d915458ef540ade6068dfe2f44e8fa733c"
```

3.5.2.5 byte

Info **[R]**

- Byte variable holds fixed-size byte array: bytes1, bytes2, ... , bytes32.
- Byte is an alias for bytes1.
- They support Comparison Operators and Bit Operators.

Address

INTEGER SIZE	DESCRIPTION
bytes1 or byte	Holds array of 1 byte.
bytes2	Holds array of 2 bytes.
bytes3	Holds array of 3 bytes.
...	...
bytes32	Holds array of 32 bytes.

Example

```
contract greeter is mortal {

  function greet() returns (byte) {
    byte   total  = 10;
    return total;
  }

}
```

output

```
Returned: "0x0a00000000000000000000000000000000000000000000000000000000000000"
```

3.5.3 Compound

Info [R]

- Compound datatypes allow for multiple items of the same type to be aggregated.
- Compound datatypes include: struct, array, string, bytes.

Data locations

- For compound types like arrays and structs, you can define "data location" whether they are stored in
  - memory by using keyword `memory` (default location for local variables)
  - storage by using keyword `storage` (default location for state variables)
- Function parameters are automatically stored in data location "calldata" which behaves like temporary memory storage.
- Assignments between storage and memory and also to a state variable always create an independent copy

3.5.3.1 bytes

Info **[R]**

- Bytes holds dynamically-sized byte array, therefore it is a special type of array.
- A bytes is similar to byte[], but it is packed tightly in call data.

Bytes

INTEGER SIZE	DESCRIPTION
bytes	Bytes holds dynamically-sized byte array, therefore it is a special type of array.

Example

```
contract greeter is mortal {

  function test() returns (bytes) {

    //Person data.
    bytes password = 'mypassword';

    //Return value.
    return password;

  }

}
```

output

```
Returned: "0x"
```

3.5.3.2 string

Info [R]

- String holds dynamically-sized UTF8-encoded string implemented as an array.
- String is equal to bytes but does not allow length or index access (for now).

String

INTEGER SIZE	DESCRIPTION
string	Holds dynamically-sized UTF8-encoded string implemented as array

Example

```
contract greeter is mortal {

  function greet() returns (string) {
    string name  = 'Ivor Online';
    return name;
  }

}
```

output

```
Returned: ""
```

3.5.3.3 struct

Info [R]

- Struct data type can contain multiple items of different type.
- Struct data type must be defined inside contract and outside member functions.
- It is not possible for a struct to contain a member of its own type but it can be the value type of a mapping member.

Example

```
contract greeter is mortal {

  //Define structure type Account.
  struct Account {
    string  username ;
    int     level;
  }

  function test(){

    //Create variable of structure type Account.
    Account ivor;
    ivor.username = 'IvorOnline';
    ivor.level    = 10;

  }

}
```

3.5.3.4 array

Info [R]

- Array data types can contain multiple items of the same type.
- Each item has numerical index starting at 0.
- Solidity supports multidimensional array as discussed later.

Example

```
function test() returns (uint) {

    //Define arrays.
    uint   [5] ages;   //Fixed    size.
    uint   [ ] levels; //Variable size.

    //Person data.
    ages   [0] = 40;   //First elements has index 0.
    levels [0] = 1;

    //Return value.
    return ages.length;

  }

}
```

output

```
Returned: "5"
```

4 Related Technologies

4.1 Chocolatey

Info [R]

- Chocolatey is application used to install other aplications.
- Chocolatey will be used to Install GETH Client.

4.1.1 Install

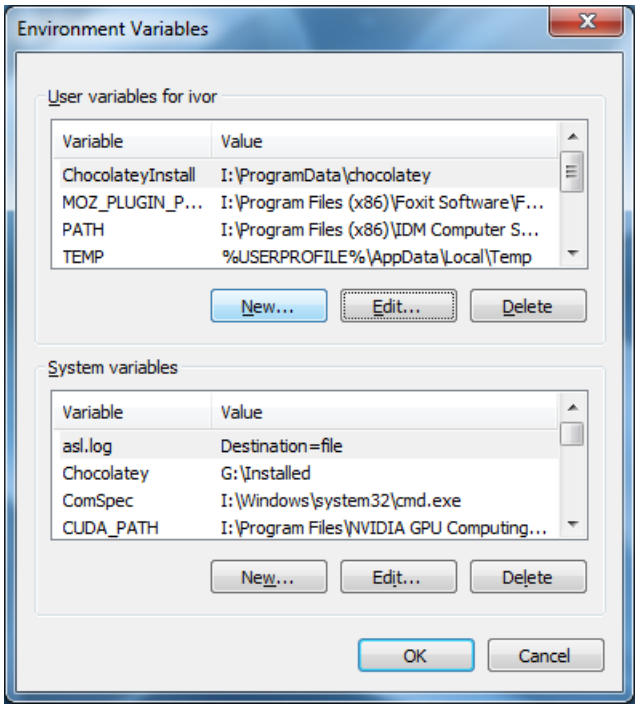
Info [R]

- This tutorial shows how to install Chocolatey either in
  - custom directory in which case you must Create Environment Variable Chocolatey as shown below
  - default directory "I:\ProgramData\chocolatey" in which case you can skip creating Environment Variable Chocolatey

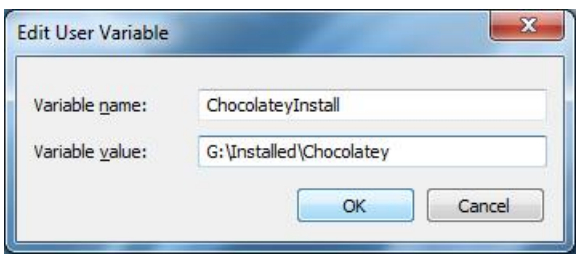
Create Environment Variable ChocolateyInstall [R]

- Show Environment Variables
- User Variables
- New...
- Variable name: **ChocolateyInstall**
- Variable value: **G:\Installed\Chocolatey**
- 3*OK

New User Variable



Specify custom directory



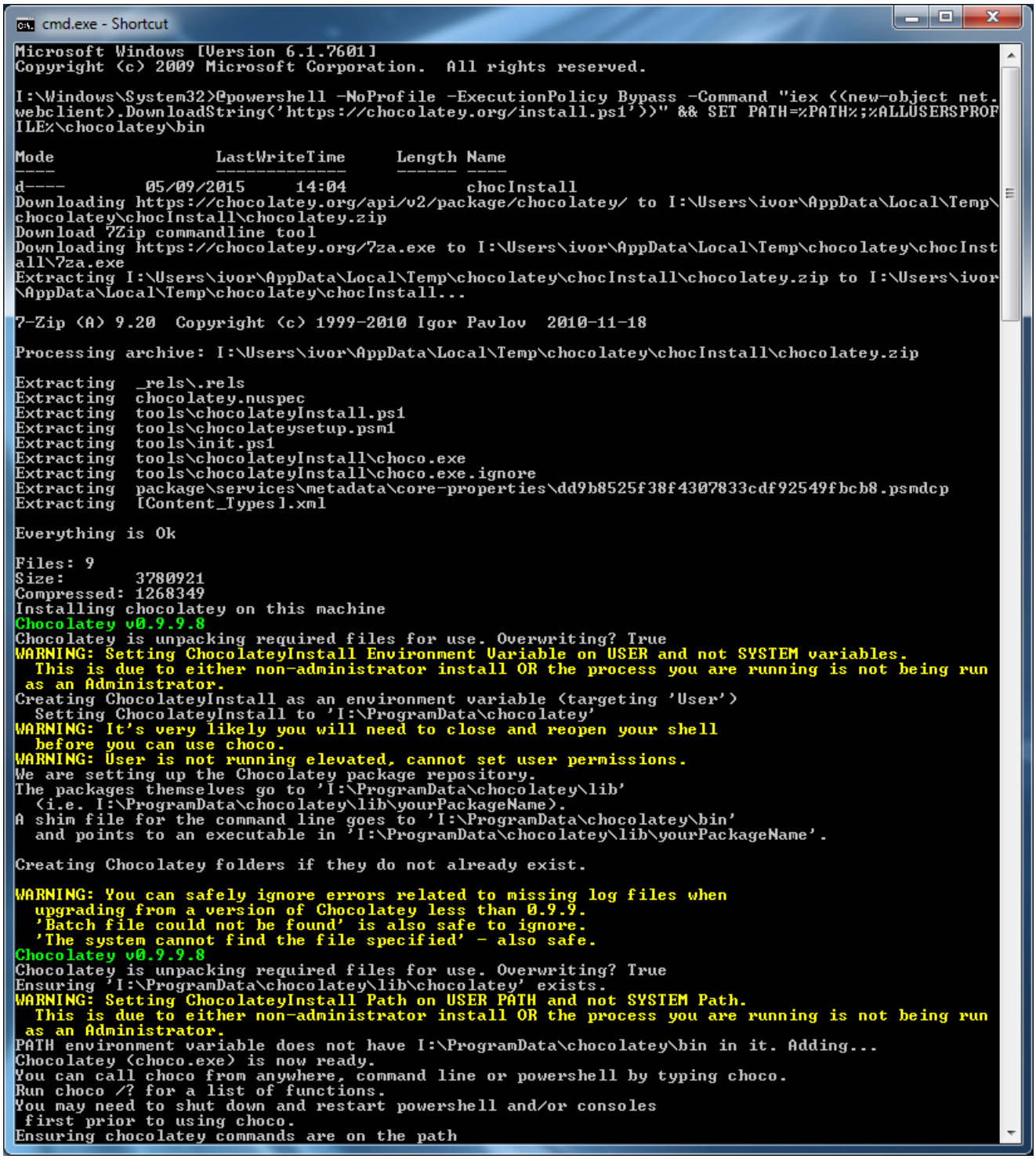
Install Chocolatey

- Start Command Prompt
- (Paste below installation command)

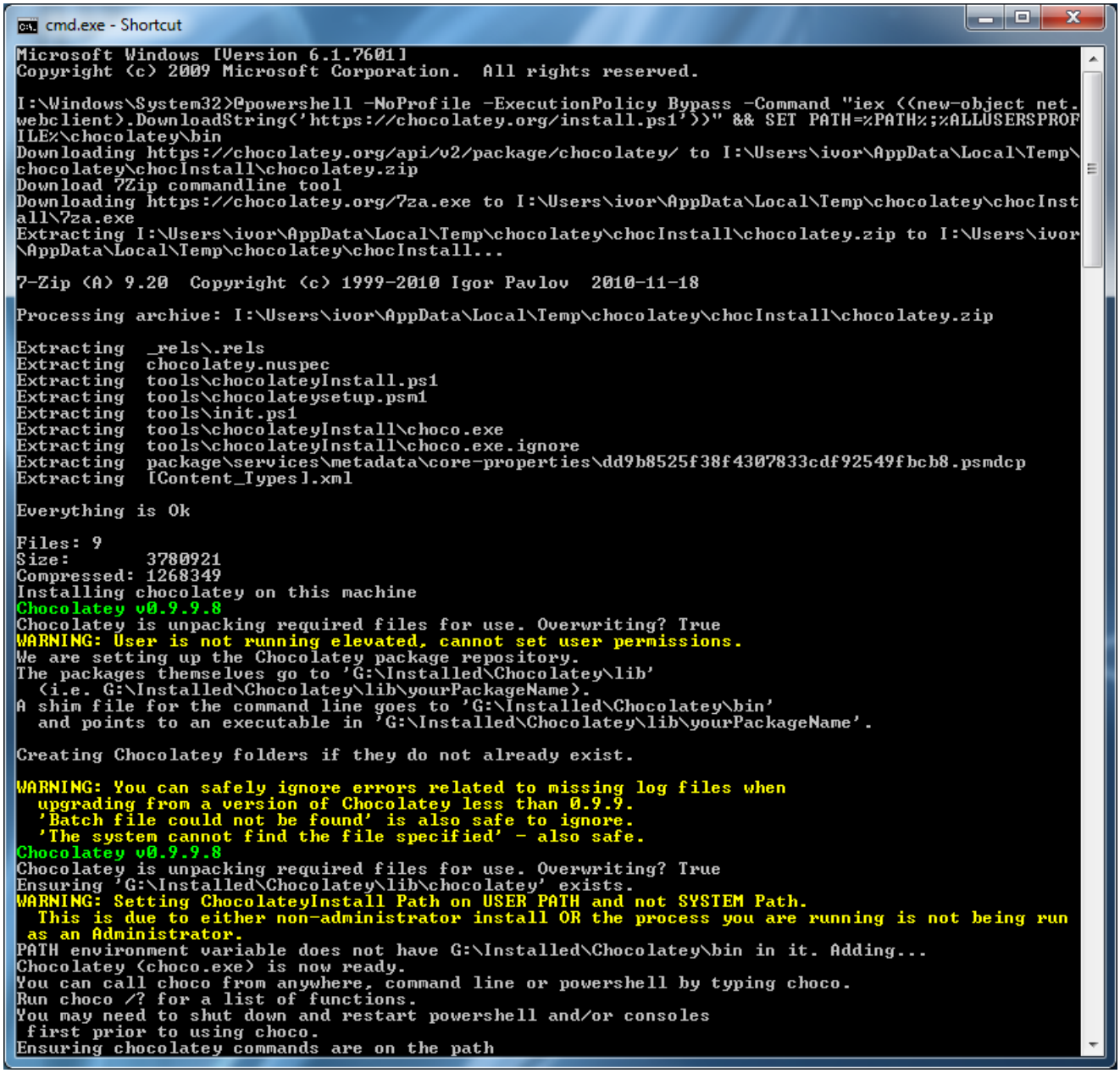
Installation command

```
@powershell -NoProfile -ExecutionPolicy Bypass -Command "iex ((new-object
net.webclient).DownloadString('https://chocolatey.org/install.ps1'))" && SET
PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin
```

Installation in default directory I:\ProgramData\chocolatey



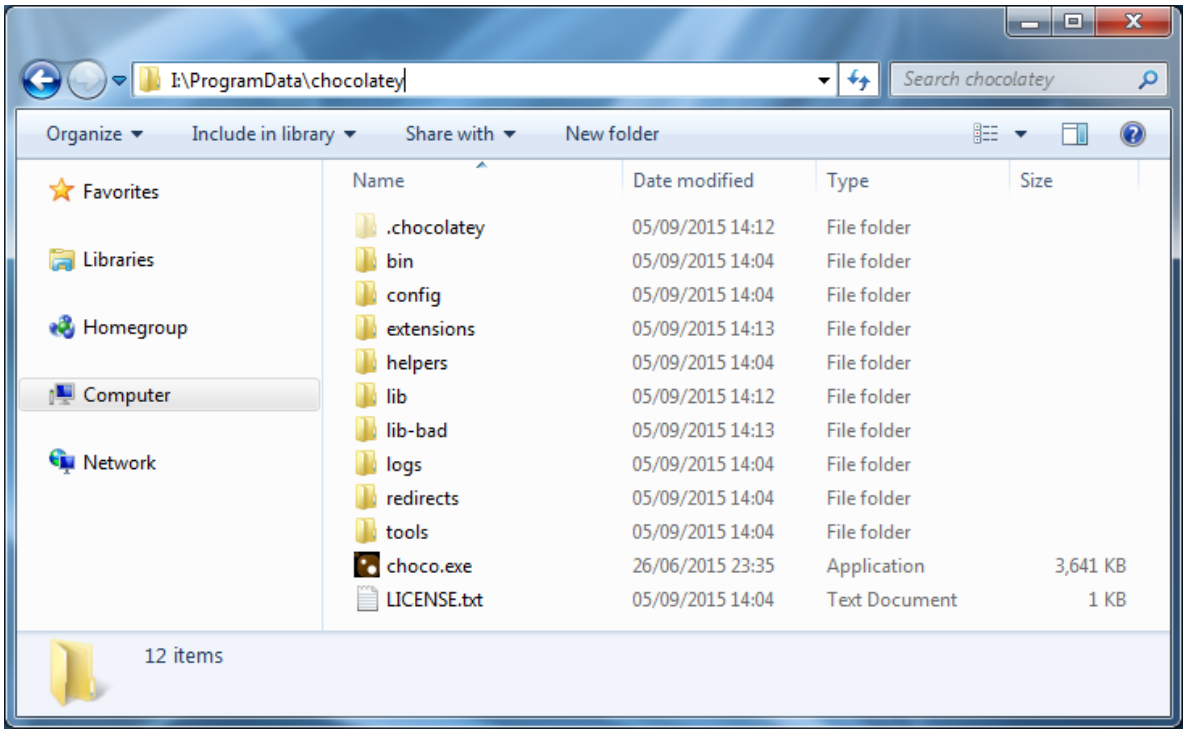
Installation in custom directory G:\Installed\Chocolatey



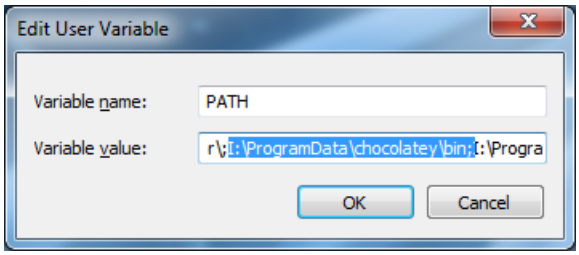
Results

- Installation will create directory structure as shown below either in default or custom directory.
- Additionally following Environment Variables are edited or created
  - PATH is edited to point to directory containing choco.exe by adding "I:\ProgramData\chocolatey\bin"
  - ChocolateyInstall is created pointing to installation directory (if you haven't created it manually as shown in first step)

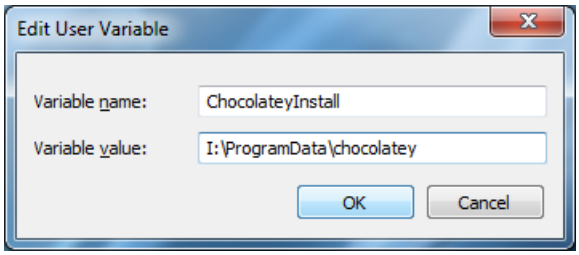
Created directory structure



Updated Environment Variable PATH



Updated Environment Variable ChocolateyInstall



Test

- Start Command Prompt
- choco

4.1.2 Uninstall

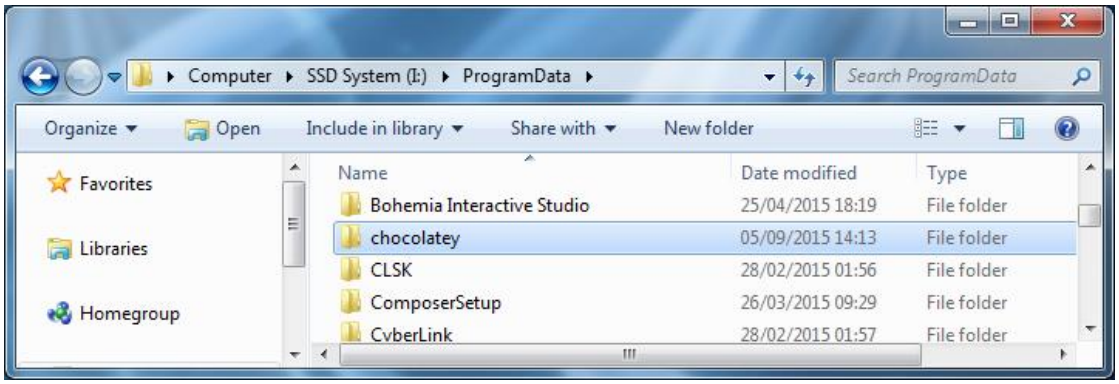
Info [R]

- This tutorial shows how to uninstall Chocolatey which is done by simply
  - deleting installation directory which will also delete applications that were installed by Chocolatey in default directory
  - removing Environment Variables if any were set

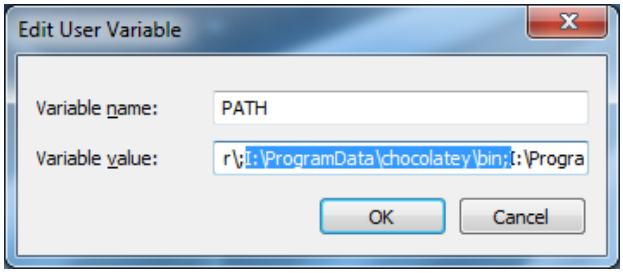
Procedure

- (Delete Installation Directory)
- Show Environment Variables
  - (Delete Environment Variable ChocolateyInstall)
  - (Remove "I:\ProgramData\chocolatey\bin" from Environment Variable PATH)

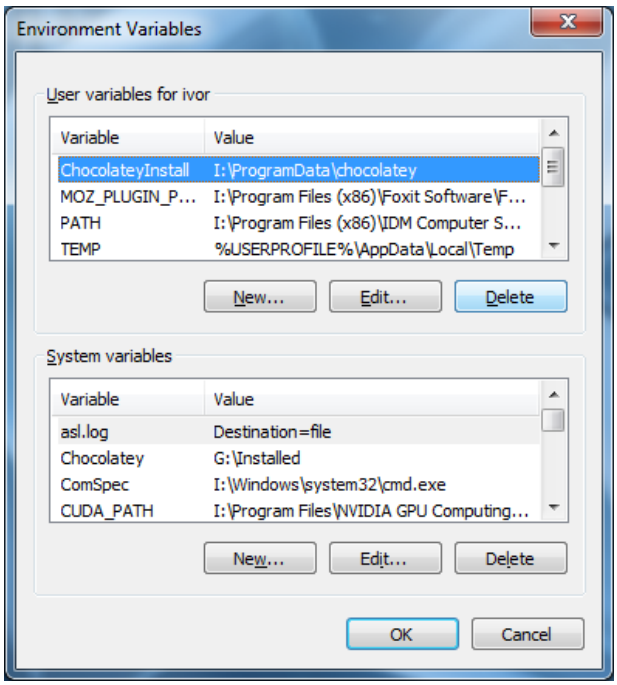
Delete Installation Directory



Delete highlighted text from Environment Variable PATH



Delete Environment Variable ChocolateyInstall



4.2 Windows

Info

- Following tutorials show how to use different Windows applications which will be needed for using Ethereum.

4.2.1 Show Environment Variables

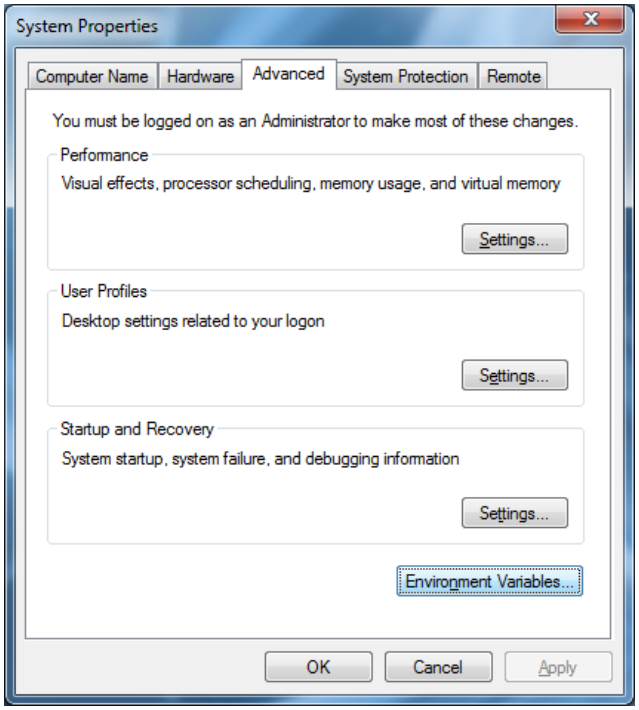
Info

- This tutorial shows show Windows Environment Variables in order to make changes to them.

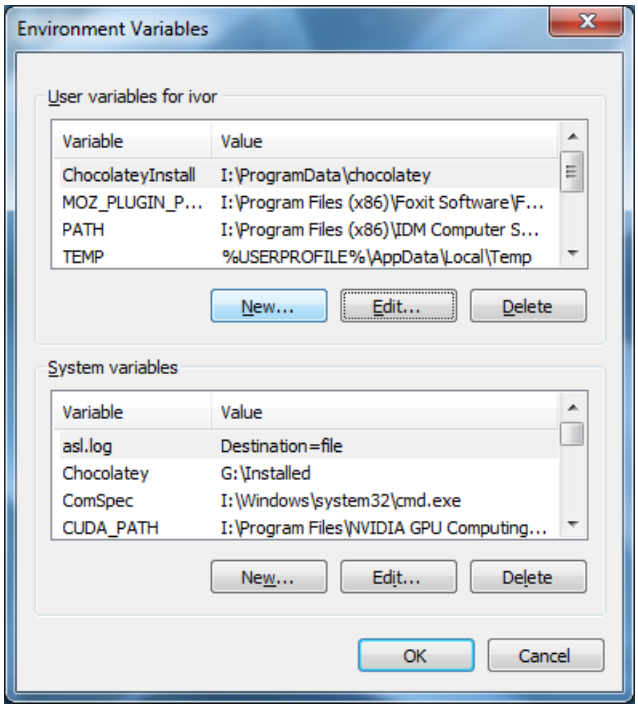
Procedure

- Start
- Control Panel
- System
- Advanced System Settings
- Tab: Advanced
- Environment Variables

Environment Variables



List of Environment Variables



4.2.2 Start Command Prompt

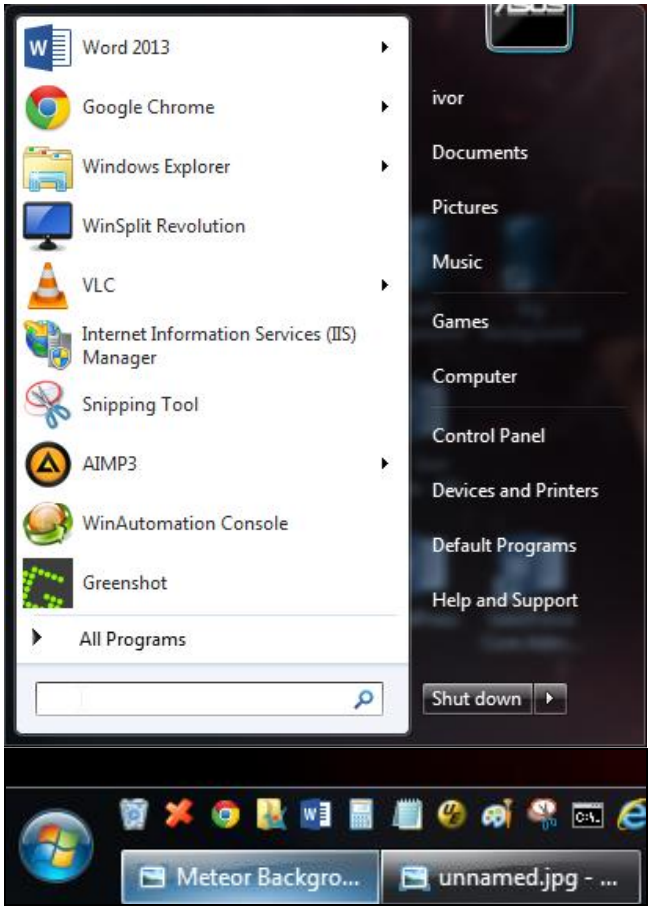
Info

- This tutorial shows how to start Command Prompt which is Windows Command Line Interface.
- To Start Command Prompt as administrator with elivated permissions use the modified second procedure shown below.

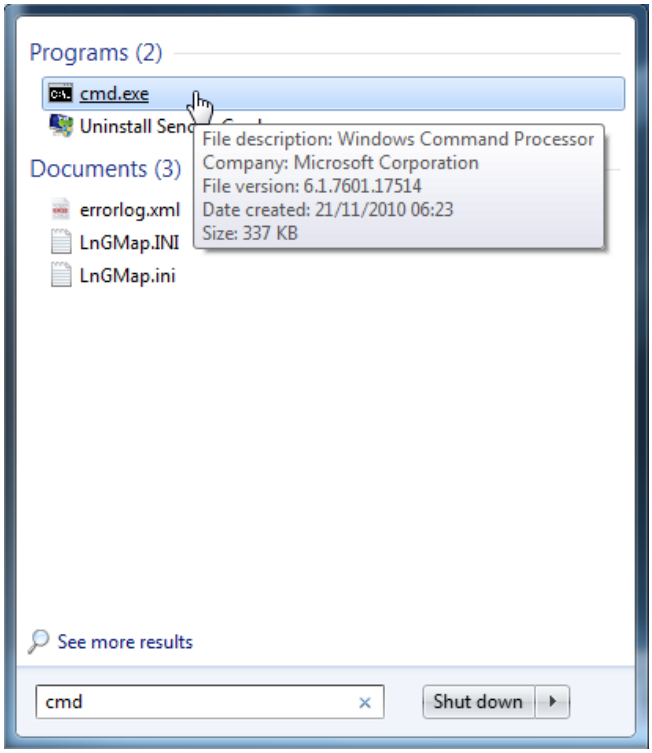
Start Normaly

- Press Windows Start button at lower left corner
- Type "cmd" in search box
- Select cmd.exe from the list

Press Start & Type "cmd" in search box



Select cmd.exe from the list



Opened Command Prompt Console

Run as administrator

- Press Windows Start button at lower left corner
- Type "cmd" in search box
- RC on cmd.exe
- **Run as Administrator**

4.3 Python

Info

- Python is programming language which is needed to run a script that generates Genesis file.

4.3.1 Install

Info

- This tutorial shows how to install Python 2.7.10 which is needed to generate Genesis file.
- You should not install Python 3.4.3 to avoid getting SyntaxError json.dumps.

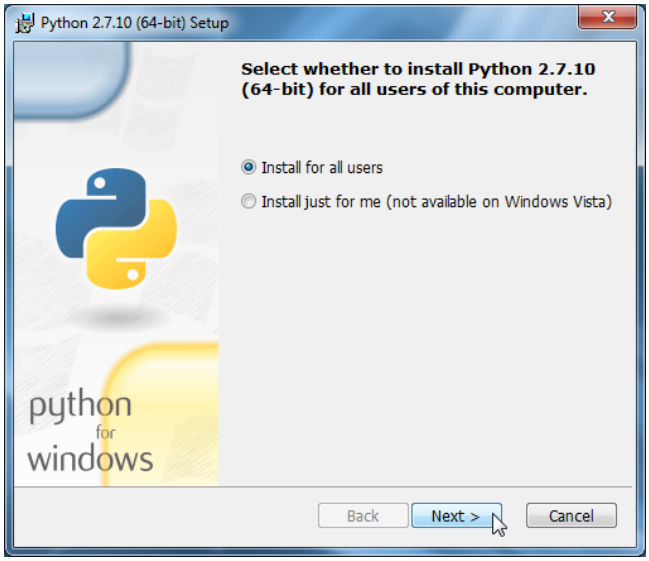
Download

- https://www.python.org/downloads/windows/
- Latest Python 2 Release - Python 2.7.10
- Windows x86-64 MSI installer
- Save as: G:\Downloads\python-2.7.10.amd64.msi

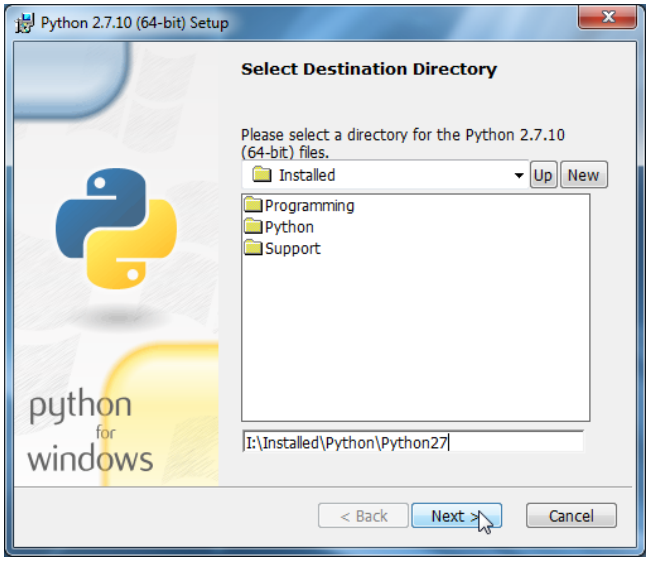
Install

- Execute G:\Downloads\python-2.7.10.amd64.msi
- Next
- Destination Directory: I:\Installed\Python\Python27
- Next
- Next
- Finish

Next



Destination Directory: I:\Installed\Python



Next

Finish

4.4 JSON Formatter

Info [R]

- This tutorial shows how to use JSON Formatter to format JSON data.
- This can be useful since Ethereum works with JSON data.

Procedure

- https://jsonformatter.curiousconcept.com/
- (Paste unformatted JSON data into JSON Data/URL box)
- Process
- (Formatted JSON appears below inside Formatted JSON Data box)

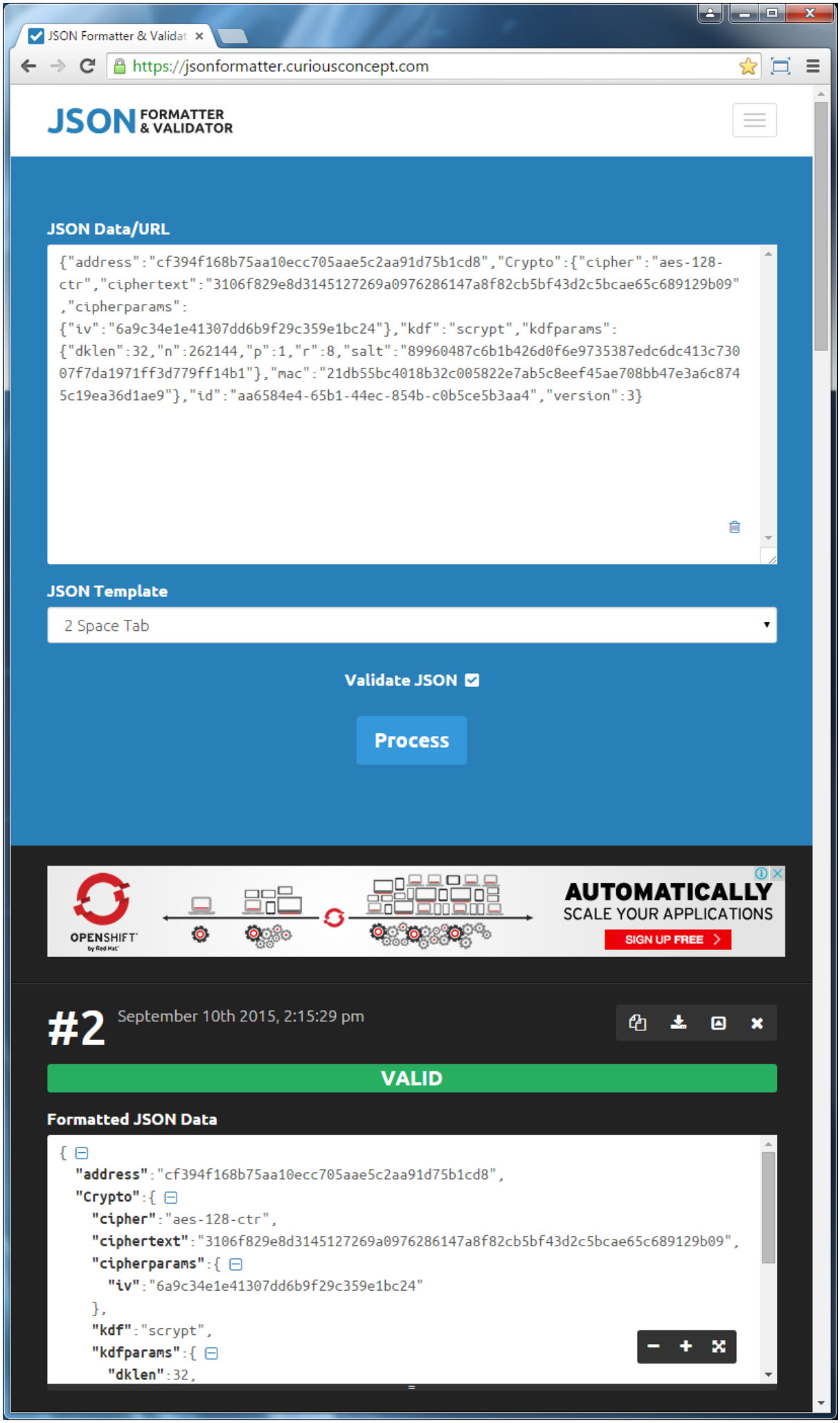
Unformatted JSON data


```
{"address":"cf394f168b75aa10ecc705aae5c2aa91d75b1cd8","Crypto":{"cipher":"aes-128-
ctr","ciphertext":"3106f829e8d3145127269a0976286147a8f82cb5bf43d2c5bcae65c689129b09","cipherparams":{"iv":"6a9c
34e1e41307dd6b9f29c359e1bc24"},"kdf":"scrypt","kdfparams":{"dklen":32,"n":262144,"p":1,"r":8,"salt":"89960487c6
b1b426d0f6e9735387edc6dc413c73007f7da1971ff3d779ff14b1"},"mac":"21db55bc4018b32c005822e7ab5c8eef45ae708bb47e3a6
c8745c19ea36d1ae9"},"id":"aa6584e4-65b1-44ec-854b-c0b5ce5b3aa4","version":3}
```


Formatted JSON data


```
{
  "address":"cf394f168b75aa10ecc705aae5c2aa91d75b1cd8",
  "Crypto":{
    "cipher":"aes-128-ctr",
    "ciphertext":"3106f829e8d3145127269a0976286147a8f82cb5bf43d2c5bcae65c689129b09",
    "cipherparams":{
      "iv":"6a9c34e1e41307dd6b9f29c359e1bc24"
    },
    "kdf":"scrypt",
    "kdfparams":{
      "dklen":32,
      "n":262144,
      "p":1,
      "r":8,
      "salt":"89960487c6b1b426d0f6e9735387edc6dc413c73007f7da1971ff3d779ff14b1"
    },
    "mac":"21db55bc4018b32c005822e7ab5c8eef45ae708bb47e3a6c8745c19ea36d1ae9"
  },
  "id":"aa6584e4-65b1-44ec-854b-c0b5ce5b3aa4",
  "version":3
}
```

JSON Formatter



5 Errors

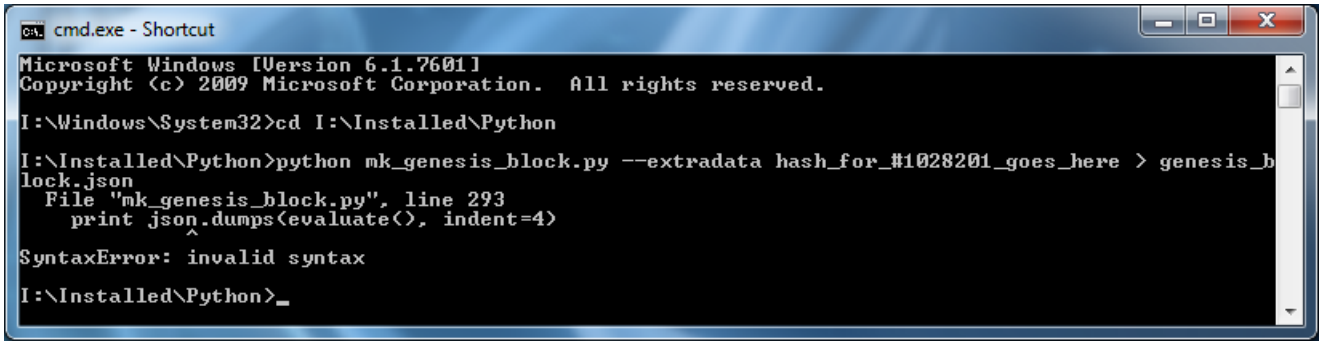
5.1 SyntaxError json.dumps

Error Message

Error Message

```
File "mk_genesis_block.py", line 293
    print json.dumps(evaluate(), indent=4)
              ^
SyntaxError: invalid syntax
```

Command Prompt



Occurrence

- This error might occur while trying to generate Genesis file as shown in Generate Genesis File after executing command

```
python mk_genesis_block.py --extradata 0x11bbe8db4e347b4e8c937c1c8370e4b5ed33adb3db69cbdb7a38e1e50b1b82fa --insigh > genesis_block.json
```

Solution

- This error occurs if you are using Python 3.x.
- Install Python 2.7.10 instead as shown in Install Python.

5.2 TypeError: 'greet' is not a function

Error Message

Error Message

```
> greeter.greet()
ReferenceError: 'greeter' is not defined
    at <anonymous>:1:1
```

Command Prompt

Occurrence

- You might get this error while using Create Contract.

Reason

- This error occurs if you are trying to use contract that hasn't been deployed already.
- If this is the case then typing just `greeter` should give you `undefined`.

Solution

- You should follow all the steps given in Create Contract.
- For instant if your contract is "waiting to be mined..." use `miner.start();` to start mining and wait for message "Contract mined! Address: 0x0aa261cc4efb737cb83492a0f85a9315d9b2b0a4"
- After that you can use `miner.stop();` and now `greeter.greet();` should work.