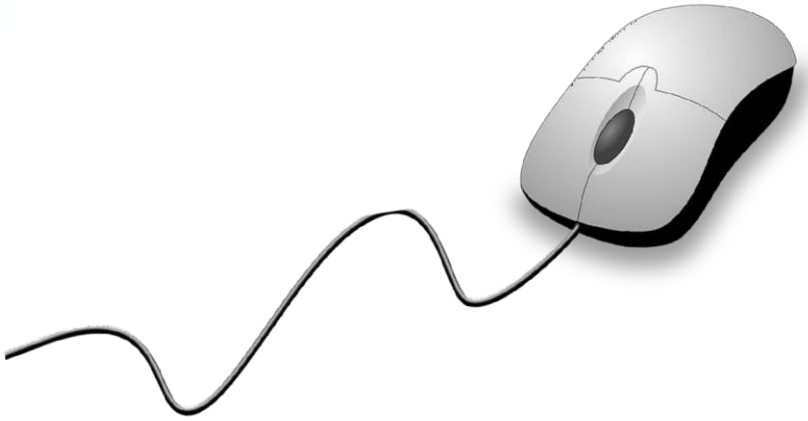
공개SW 솔루션 설치 & 활용 가이드

기타 > 블록체인



ethereum

제대로 배워보자

How to Use Open Source Software

Open Source Software Installation & Application Guide

OPENUP

오픈소스 소프트웨어 통합지원센터

Open Source Software Support Center

CONTENTS

1. 개요

소개	• Ethereum은 블록체인 기술을 기반으로 스마트 계약 기능을 구현하기 위한 분산 컴퓨팅 플랫폼 • 이더리움이 제공하는 이더(Ether)는 비트코인과 마찬가지로 암호화폐의 일종으로 거래 • 이더리움의 화폐 단위는 ETH로 표시 • 가장 대표적인 알트코인		
주요기능	• 유연한 아키텍처를 채택하여 단일 API를 통해 데스크톱, 서버 또는 휴대기기에 장착된 하나 이상의 CPU 또는 GPU에 연산 배포 가능		
대분류	• 기타	**소분류**	• 블록체인
라이선스 형태	• GNU LGPL v3	**사전설치 솔루션**	• Go, C++, Rust
실행 하드웨어	• X86 , ARM 호환 프로세서, • 램 1GB 이상의 하드웨어 • 500MB 이상의 디스크 공간 (최소 256MB)	**버전**	• 1.8.17 (2018년 10월 기준)
특징	• 블록체인에 화폐 거래 기록뿐 아니라 계약서 등의 추가 정보를 기록할 수 있다는 점에 착안하여 전 세계 수많은 사용자들이 보유하고 있는 컴퓨팅 자원을 활용해 분산 네트워크를 구성하고, 이 플랫폼을 이용하여 SNS, 이메일, 전자투표 등 다양한 정보를 기록하는 시스템 창안		
보안취약점	• 취약점 ID : CVE-2018-12018 • 심각도 : 7.5 HIGH(V3) • 취약점 설명 : 1.8.11 이전의 Go Ethereum (일명 geth)의 LES 프로토콜 구현에서 GetBlockHeadersMsg 처리기는 배열 인덱스에 대한 정수 signedness 오류로인해액세스위반이발생 가능, 공격자가 -1 쿼리로 패킷을 보내서 서비스 거부 공격을 시작 가능, 취약한 원격 노드는 즉각 EPoD (Ethereum Packet of Death) 문제와 같은 공격에 의해 문제 • 대응방안 : 최신 패치 버전 사용 • 참고 경로 : https://github.com/ethereum/go-ethereum/pull/16891		
개발회사/커뮤니티	• Etheeum Foundation https://ethereum.org/foundation		
공식 홈페이지	• https://www.ethereum.org/		

2. 기능요약

- Etherium 의 주요 기능

주요기능	지원여부
32 / 64비트 OS	32 / 64 지원 (x86 / x86_64), ARM
Type	Decentralized computing, Blockchain, Cryptocurrency
Anaconda 설치	지원
Source 저장소	https://github.com/ethereum
결함추적시스템	https://github.com/ethereum/go-ethereum/issues
언어지원	English, 중국어, 일본어, 한국어
지원 OS	Linux, Windows, macOS, POSIX, Raspbian

3. 실행환경

- 하드웨어 제약이 거의 없음
- 다양한 개발 언어별 클라이언트 지원

Client	Language
Go-Ethereum	go
Parity	Rust
Cpp-Ethereum	C++
Pyehapp	Python
Ehereumjs-lib	Javascript
Etherum(j)	Java
Ruby-Ethereum	Ruby
ehereumH	Haskell

4. 설치 및 실행

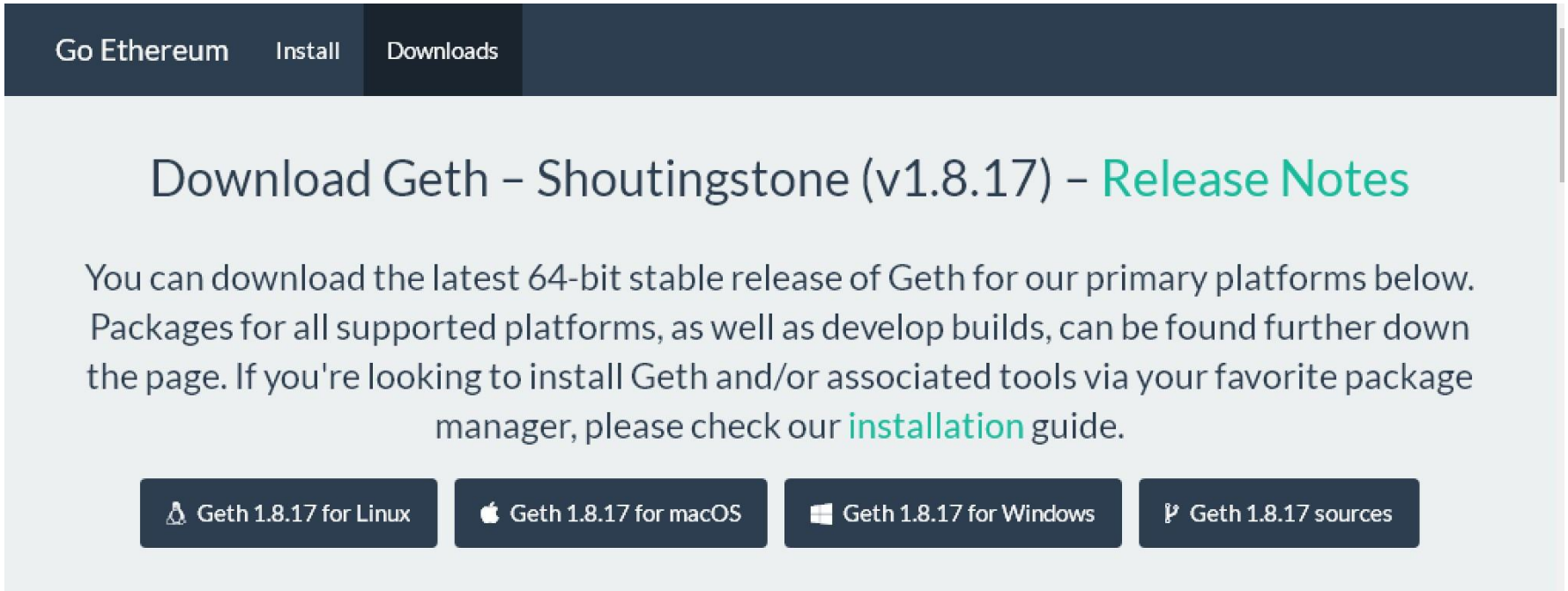
세부 목차

1. 설치 이미지 준비
2. 설치
3. 초기 설정 및 계정 설정
4. 설치 진행
5. 설치 완료

4. 설치 및 실행

4.1 설치 이미지 준비

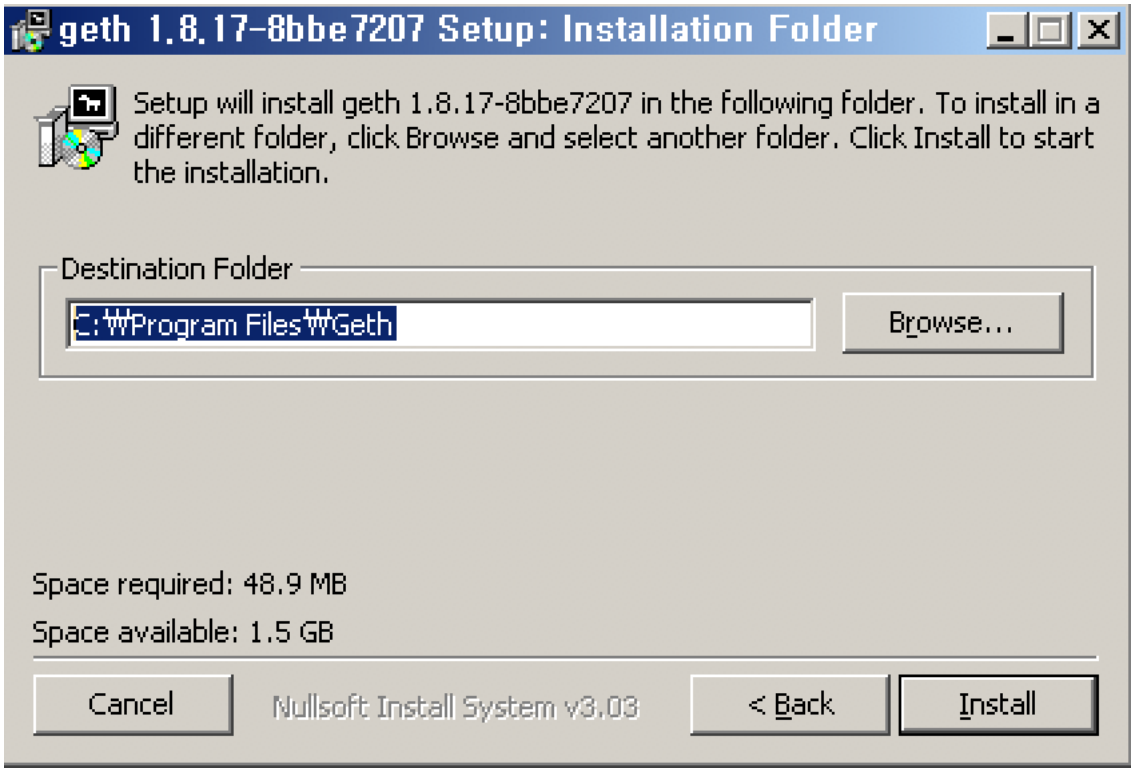
- 실제 개발자들은 리눅스 등의 환경을 사용할 수 있겠지만, 가장 쉽게 접근할 수 있는 것이 윈도우 환경에서 설치임
- Geth 파일을 다운로드, 다운로드는 이더리움 다운로드 웹사이트 이용
  - 이더리움 다운로드 : https://geth.ethereum.org/downloads/



4. 설치 및 실행

4.2 설치

- [Geth 1.8.17 for Windows] 다운로드
- 압축파일 형태이므로 실행 시 폴더를 작업하기 쉬운 곳으로 위치하면 좋음
- D:\Geth\ 폴더 안에 압축 풀기
- 압축을 풀면 여러 실행파일들이 Geth 폴더 안에 생성되는 것 확인



4. 설치 및 실행

4.3 초기 설정 및 계정 설정 (1/2)

- 초기 설정은 진행하지 않아도 되지만, 설정을 원할 경우 genesis.json 파일을 만들어 초기값을 넣을 수 있음
- Geth 1.8.17 버전의 genesis.json 파일의 내용은 아래와 같음

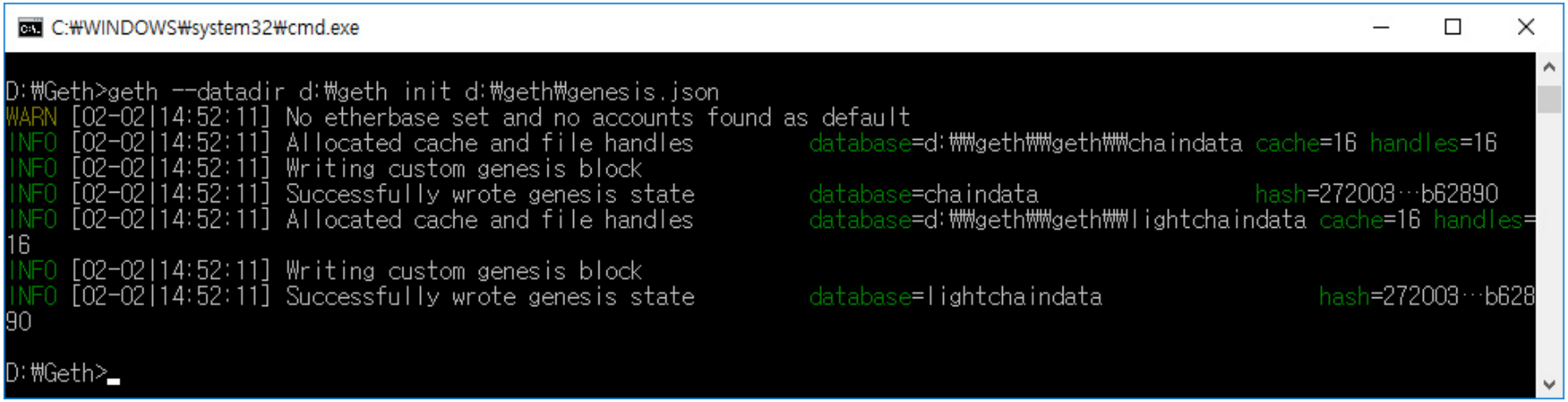
```
{
  "config": {
        "chainId": 0,
        "homesteadBlock": 0,
        "eip155Block": 0,
        "eip158Block": 0
    },
  "alloc"      : {},
  "coinbase"   : "0x0000000000000000000000000000000000000000",
  "difficulty" : "0x20000",
  "extraData"  : "",
  "gasLimit"   : "0x2fefd8",
  "nonce"      : "0x0000000000000042",
  "mixhash"    : "0x0000000000000000000000000000000000000000000000000000000000000000",
  "parentHash" : "0x0000000000000000000000000000000000000000000000000000000000000000",
  "timestamp"  : "0x00"
}
```

4. 설치 및 실행

4.3 초기 설정 및 계정 설정(2/2)

- 설정파일은 버전마다 조금씩 다르기 때문에 github.com 등에서 찾음
- Genesis.json 파일에 저장한 후
- 사용하고자 하는 데이터 폴더에 넣어두고 아래와 같은 명령어로 초기화 시킴

  - d:\Geth\geth –datadir D:\Geth init d:\Geth\genesis.json

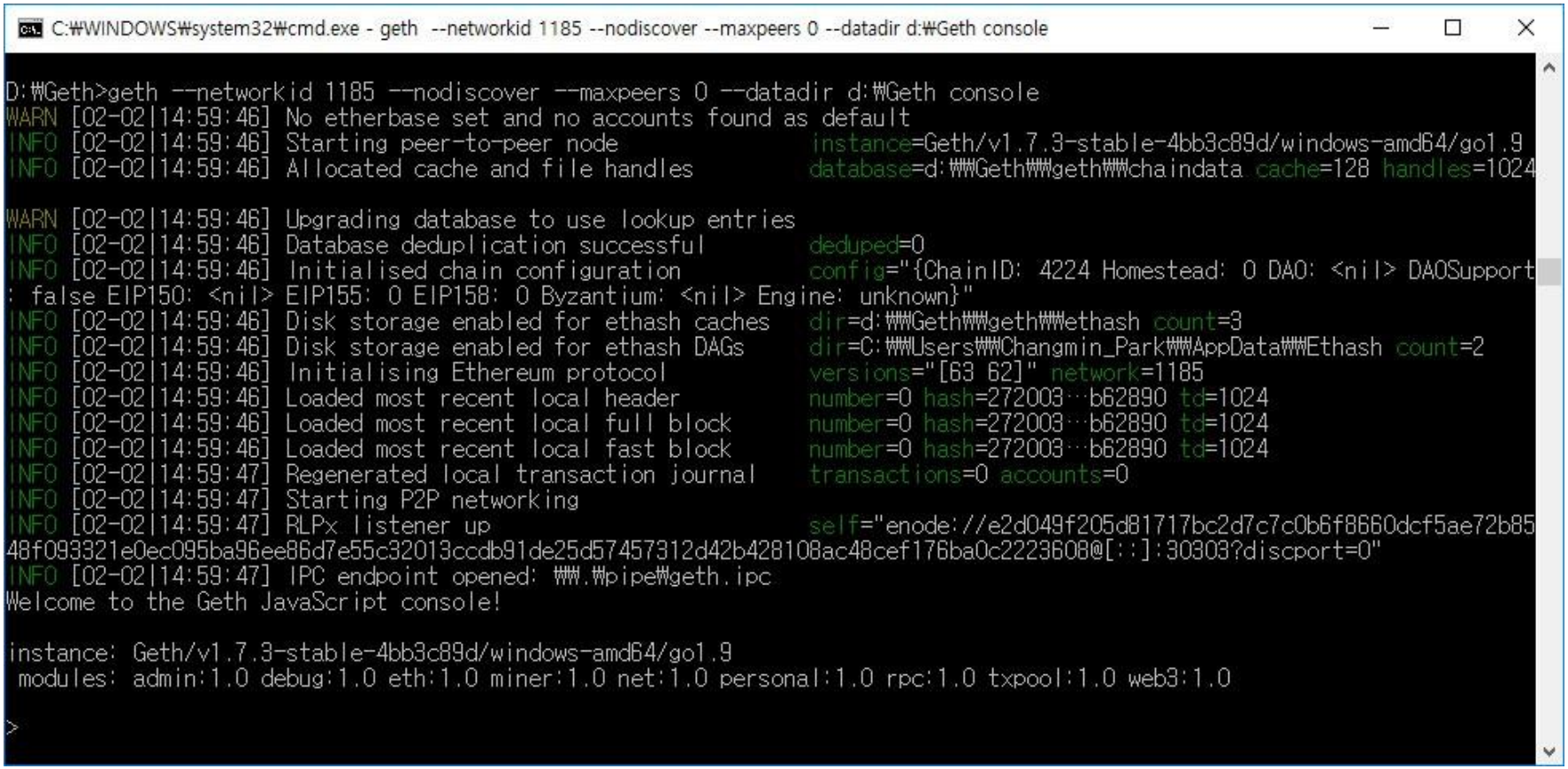


4. 설치 및 실행

4.4 설치 실행(1/3)

- 정상적으로 초기화 되었다면 d:\Geth 폴더 안에 geth, keystone 등의 폴더가 생성
- Geth 실행

  - d:\Geth\geth –networkid 1185 –nodiscover –maxpeers 0 –datadir d:\Geth console



4. 설치 및 실행

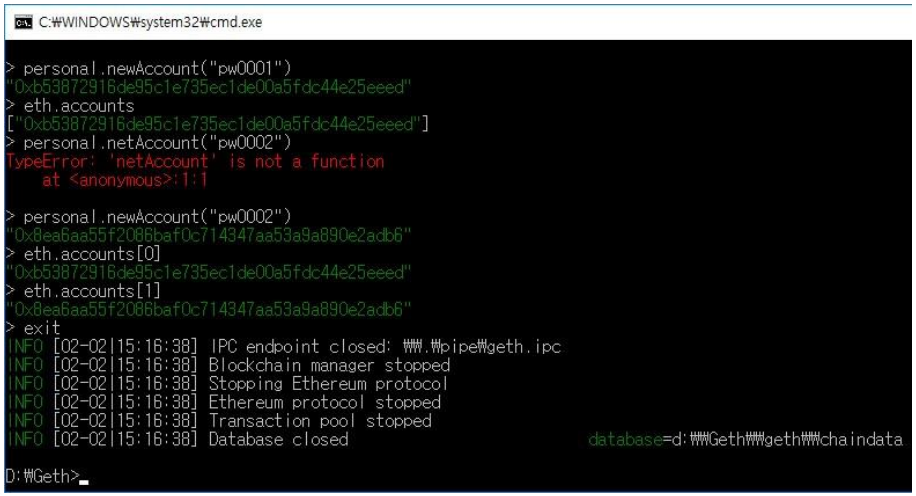
4.4 설치 실행(2/3)

- 각 명령어는 아래와 같은 의미를 가짐
  - networkid 1185 : 네트워크를 식별할 수 있는 id의 생성으로, 0~3을 제외한 임의의 정수를 사용, 여기서는 1185라는 숫자를 씀. 각자 다른 숫자를 이용해 테스트
  - nodiscover : 생성자의 노드를 다른 노드에서 검색할 수 없게 하는 옵션
  - maxpeers 0 : 생성자의 노드에 연결할 수 있는 노드의 수임, 0을 지정하면 다른 노드와 연결하지 않음
  - datadir d:\Geth : 데이터 폴더를 지정하는 것으로, 지정하지 않으면 기본 폴더 사용
  - console : 실행 후 자바스크립트 콘솔 기동

4. 설치 및 실행

4.4 설치 실행(3/3)

- 명령어를 입력한 후 > 표시의 프롬프트가 떴다면 정상적으로 실행된 것임
- 계정(account) 생성
- 계정에는 EOA(Externally Owned Account)와 Contract 두가지 종류가 존재
- EOA는 일반 계정으로 Ether를 송금하거나 계약을 실행할 수 있음
- Contract 계정은 계약을 블록체인에 배포할 때 만들어지는 계정으로 블록체인에 존재하며, 다른 계정으로 메시지를 수신해 코드를 실행할 수 있음
- EOA 계정을 만들 때는 personal.newAccount("비밀번호") 명령어 사용
  - personal.newAccount("pw0001")



4. 설치 및 실행

4.4 설치 완료

- exit 명령어로 콘솔을 빠져나올 수 있으며, Geth 프로세스도 함께 종료
- D:\Geth 폴더로 이동하여 확인해보면 keystore 폴더 안에 2개의 파일이 생성된 것 확인

5. 기능소개

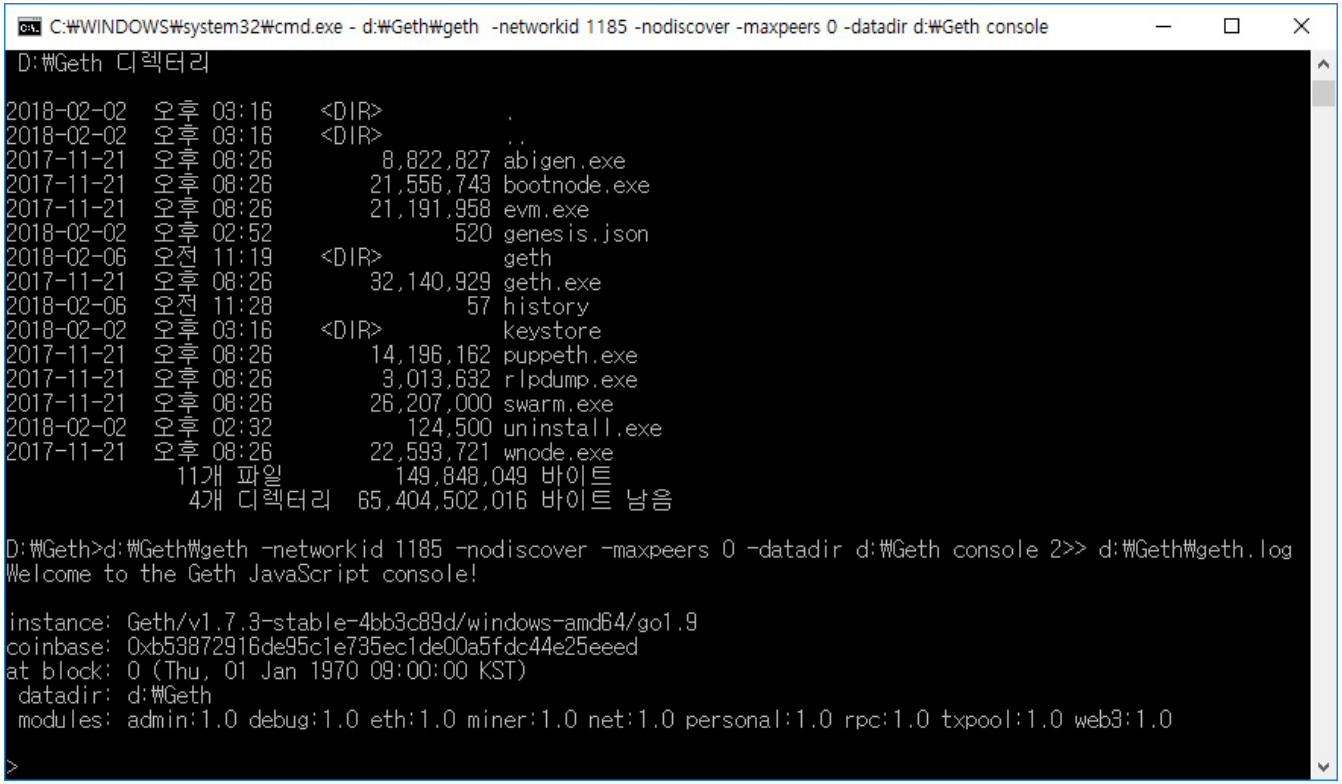
세부 목차

1. 채굴
2. 송금
3. 수수료

5. 기능소개

5.1 채굴(1/4)

- Geth를 구동, 지난 번에 했던 방법과 동일하게 구동하며, 대신 log를 남기는 명령어 추가
  - 로그를 남기는 명령어로 실행하지 않으면, 채굴 중 그 내용이 모두 화면에 표시되기 때문에 추가 작업을 하기에 다소 번거로울 수 있음
  - >d:\Geth\geth--networkid 1185--nodiscover--maxpeers 0--datadir d:\Gethconsole 2>>d:\Geth\geth.log



5. 기능소개

5.1 채굴(2/4)

- 채굴을 했을 때 보상받는 계정을 Etherbase라고 하며, 이것은 기본적으로 첫번째 생성된 계정으로 지정되고, eth.coinbase 명령어로 확인 가능

```
> eth.coinbase
```

- 물론, miner.setEtherbase 명령어로 계정의 변경도 가능하니 참고, 계정을 만들었으니, 채굴 전에 잔액을 확인하는 방법 먼저 알아두는 것이 좋음
- 잔액 확인은 eth.getBalance() 함수로 가능, 현재 잔액이 없으므로 0으로 확인

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

5. 기능소개

5.1 채굴(3/4)

- 채굴은 miner.start(쓰레드 수)로 시작, 쓰레드 수는 실행하는 쓰레드의 개수를 입력하면 되고, 이번에는 하나의 쓰레드만 가동

  >miner.start(1)
  null

- 채굴 중인 내용은 시작할 때 지정한 geth.log 파일을 통해서 확인할 수 있고, eth.mining 명령으로 실행 중인지 확인, 또 eth.hashrate와 eth.blockNumber 등의 명령어로 진행된 채굴 상황을 볼 수 있음

  >eth.mining
  true
  > eth.hashrate
  57032
  >eth.blockNumber
  166

5. 기능소개

5.1 채굴(4/4)

- 블록이 생성되었다면 miner.stop() 명령어로 채굴을 중지한 후, 잔액 확인
- eth.getBalance() 명령을 실행하면 wei 단위로 표시되기 때문에 엄청난 잔액이 있는 것처럼 보일 것임
- 1ether는 10의 18승 wei 임
- 이것을 ether로 보기 위해서는 web3.fromWei() 명령어 이용
- 현재 채굴로 받을 수 있는 보상은 1블록에 5ether임
- 채굴을 중단할 때까지 블록의 길이가 178이었으므로 178×5=890이 되는 것임

```
>eth.getBalance(eth.accounts[0])
8900000000000000000
>web3.fromWei(eth.getBalance(eth.accounts[0]),"ether")
890
>eth.blockNumber
178
```

5. 기능소개

5.2 송금(1/3)

- 송금 명령어는 eth.sendTransaction() 명령어로 가능, 하지만 실행하면 오류가 발생
- 다음 명령어를 실행해, accounts[0]에서 accounts[1]으로 10ether 송금

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

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

Error: authentication needed: password or unlock
    at web3.js:3143:20
    at web3.js:6347:15
    at web3.js:5081:36
    at <anonymous>:1:1

>
```

5. 기능소개

5.2 송금(2/3)

- 오류가 발생하는 이유는 송금에는 수수료가 발생하기 때문이며, 실수로 다른 계정에서 송금하는 것을 방지하기 위해 오류를 만드는 것임
- 이럴 때는 계정의 잠금을 풀고 송금을 진행할 수 있음
- 계정 잠금을 푸는 명령어는 아래와 같음

  > personal.unlockAccount(eth.accounts[0],”pw0001″)

- unlockAccount() 함수의 변수는 첫번째가 계정, 두번째가 그 계정의 비밀번호
- 계정만 입력하게 되면 암호 입력창이 나타나 입력
- 잠금 해제는 보통 300초 동안 유지되며, 필요에 따라 마지막에 추가 변수(초 단위)를 입력하여 해제 시간 조절

```
> personal.unlockAccount(eth.accounts[0],"pw0001")
true
> eth.sendTransaction({from:eth.accounts[0], to:eth.accounts[1], value:web3.toWei(10,"ether")})
"0xa0693e6a344e3a856fa3a6014fb7a11ae4168ace95ce05c2825f1a0cc1e6fcb9"
>
```

5. 기능소개

5.2 송금(3/3)

- 잠금을 해제한 후 송금 명령어를 다시 입력하면 트랜잭션 ID가 화면에 표시되고 송금진행
- 하지만, 현재 상황에서 실제 accounts[1]에는 잔액이 확인되지 않음
  - 그것은 아직 블록체인에 송금 내역이 기록되지 않았기 때문임
- 대신, 아직 보류 중인 트랜잭션을 확인하게 되면 송금된 내역을 확인할 수 있음, 보류 중인 트랜잭션은 아래의 명령어로 확인 가능

  > eth.pendingTransactions

```
> eth.getBalance(eth.accounts[1])
0
> eth.getBalance(eth.accounts[0])
1.57e+21
> eth.pendingTransactions
[{
    blockHash: null,
    blockNumber: null,
    from: "0xfc92bdbc03860e782d6b859236666c4c4a781e53",
    gas: 90000,
    gasPrice: 18000000000,
    hash: "0xa0693e6a344e3a856fa3a6014fb7a11ae4168ace95ce05c2825f1a0cc1e6fcb9",
    input: "0x",
    nonce: 0,
    r: "0xc106b94585cbfb6374d6bbd61140abc0c01f47c39ee125d62ab7e74a86f46d7c",
    s: "0x656dbfcce988304d18b24316032608d222802af599d3bc4c9f6599fa728dc708",
    to: "0xd1ec822bed39ba51a59471b7fd92c2bff405ff92",
    transactionIndex: 0,
    v: "0x2123",
    value: 1000000000000000000
}]
>
> miner.start(1)
null
> eth.pandingTransactions
undefined
> eth.pendingTransactions
[]
> miner.stop()
true
> _
```

5. 기능소개

5.3 수수료(1/3)

- account[1]에서 account[2]로 5 이더를 전송하는 작업 진행
- Account[2]가 없다면 먼저, personal.newAccount() 명령어를 통해 생성
- 먼저 account[1]의 잠금을 해제하고, sendTransaction()을 통해 5 이더 보냄
- 그 후, miner.start(1)로 블록을 생성하면 트랜잭션이 처리됨

```
> personal.unlockAccount(eth.accounts[1],"pw0002")
true
> eth.sendTransaction({from:eth.accounts[1], to:eth.accounts[2], value:web3.toWei(5,"ether")})
"0x8a3f29782c109c92335bcda2ac9763eb9d1506c54c463d094ce35a65639d6249"
> miner.start(1)
null
> eth.pendingTransactions
[]
> miner.stop()
true
> eth.getBalance(eth.accounts[2])
5000000000000000000
> eth.getBalance(eth.accounts[1])
4999622000000000000
> eth.getTransaction("0x8a3f29782c109c92335bcda2ac9763eb9d1506c54c463d094ce35a65639d6249")
{
  blockHash: "0xd6ea793403d93d9310c47c9d2f66000dddb1659ab0a94b395aeef8e37f3c2361",
  blockNumber: 348,
  from: "0xd1ec822bed39ba51a59471b7fd92c2bff405ff92",
  gas: 90000,
  gasPrice: 18000000000,
  hash: "0x8a3f29782c109c92335bcda2ac9763eb9d1506c54c463d094ce35a65639d6249",
  input: "0x",
  nonce: 0,
  r: "0x8649f322d1ea33d7b645ad442b86c6552cdd0613e965e347df05c41bab8975a",
  s: "0x5de2d133646e2ca9b37ec427b6596d3f6834faddd29ba5f8ef67f39241652a9c",
  to: "0x70cf6d70b3c41009ad34f5c3a1811f1da48c0fab",
  transactionIndex: 0,
  v: "0x2123",
  value: 5000000000000000000
}
>
```

5. 기능소개

5.3 수수료(2/3)

- 작업 후 각 account의 잔고를 확인해보면, account[2]에 5 이더가 들어간 것 확인
- 하지만, account[1]에는 잔고가 4.999622 이더가 남아 있는 것 확인
- 이것은 송금이 발생된 후에 수수료인 gas가 보낸 계정에서 자동 출금되었기 때문임
- gas의 양을 확인하기 위해 getTransaction() 명령어로 확인

  > eth.getTransaction("0x8a3f29782c109c92335bcda2ac9763eb9d1506c54c463d094ce35a65639d6249")

5. 기능소개

5.3 수수료(3/3)

- 트랜잭션 ID를 입력하여 결과값을 확인하면 gas와 gasPrice 값이 표시됨
- gas의 값은 지불할 수 있는 최대 수수료를 의미하고, 실제 지불한 gas는 아님
- 실제 지불한 gas는 잔액의 차이를 gasPrice(wei 단위)로 나눈 값임
- 그러므로 현재 지불한 gas는 아래의 공식으로 계산됨
  - 잔액의 차액은 318,000,000,000,000 wei
  - 지불한 수수료 = 318,000,000,000,000 / 18,000,000,000 = 17,667 gas
  - 트랜잭션에 표시한 90,000 gas에 비해서 적은 양이 수수료로 사용된 것 확인

6. 활용예제

세부 목차

1. Smart Contract 환경
2. 개발 및 배포
3. 가나슈 설치
4. 가나슈 실행
5. Remix 접속
6. 스마트 컨트랙트 예제 실행

6. 활용예제

6.1 Smart Contract 환경

- **개발 도구 설명**

- IDE

통합 개발 환경이라고 하며 개발하는 언어에 따라서 사용하는 IDE가 달라짐. C/C++의 경우에는 Visual Studio, Java는 Eclipse, IntelliJ IDEA, Python PyCharm 등 여러가지 IDE가 존재함. 윈도우상에 설치해서 사용하는 IDE가 있는 반면 그냥 웹 브라우저 상에서 동작하는 IDE들도 존재함. 우리는 우선 별도의 설치 없이 사용할 수 있는 Online IDE인 Remix 사용

- Ethereum Node / Network

보통은 Geth, Parity 등 이용해서 블록을 모두 동기화 시키고 실제 이더리움 메인/테스트 네트워크에 접속해서 진행. 하지만 블록을 동기화만 해도 2~3일 정도 걸리고(약 한 달 전 제가 동기화 시킨 기준), 블록을 채굴하기까지 기다려야 되는 등 개발 중에 불편한 점이 많음. 그래서 개발 중에는 Ganache와 같은 가상 혹은 프라이빗 네트워크 상에서 스마트 컨트랙트 구동해 보고 테스트넷을 거쳐 메인넷에 올라 가게됨

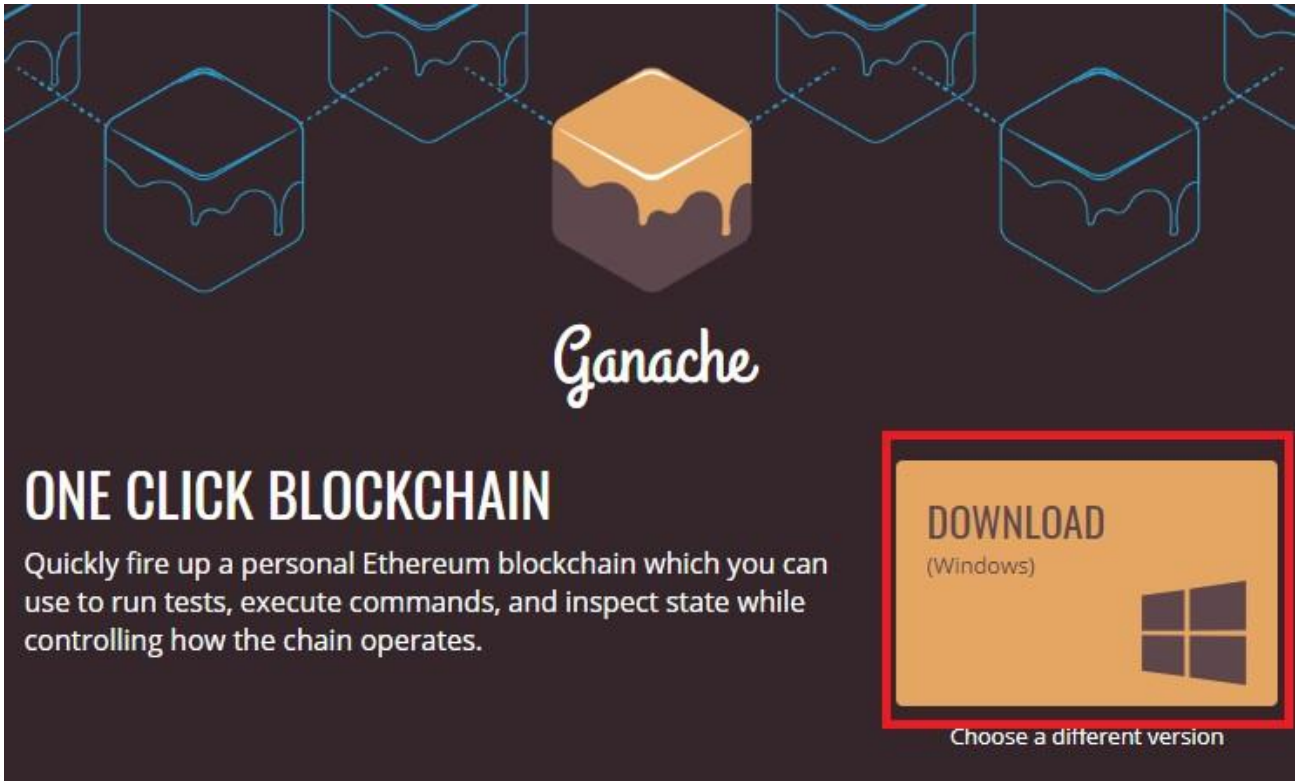
- **개발 및 배포 과정**
  - 이더리움 네트워크 개발은 개발용 testRPC에서 개발하고 검증(TestNet)을 한 후 시스템이 어느 정도 안정화 되었다면 MainNet으로 서비스 함

* TestRPC -> TestNet -> MainNet
* TestRPC : 개발을 위한 이더리움 개발 네트워크를 구성하고 개발하는 과정
* TestNet : 개발 완료한 후 MainNet과 동일한 환경에서 테스트하는 과정
* MainNet : 실제 서비스에 사용할 수 있도록 배포하는 과정

6. 활용예제

6.3 가나슈 설치(1/3)

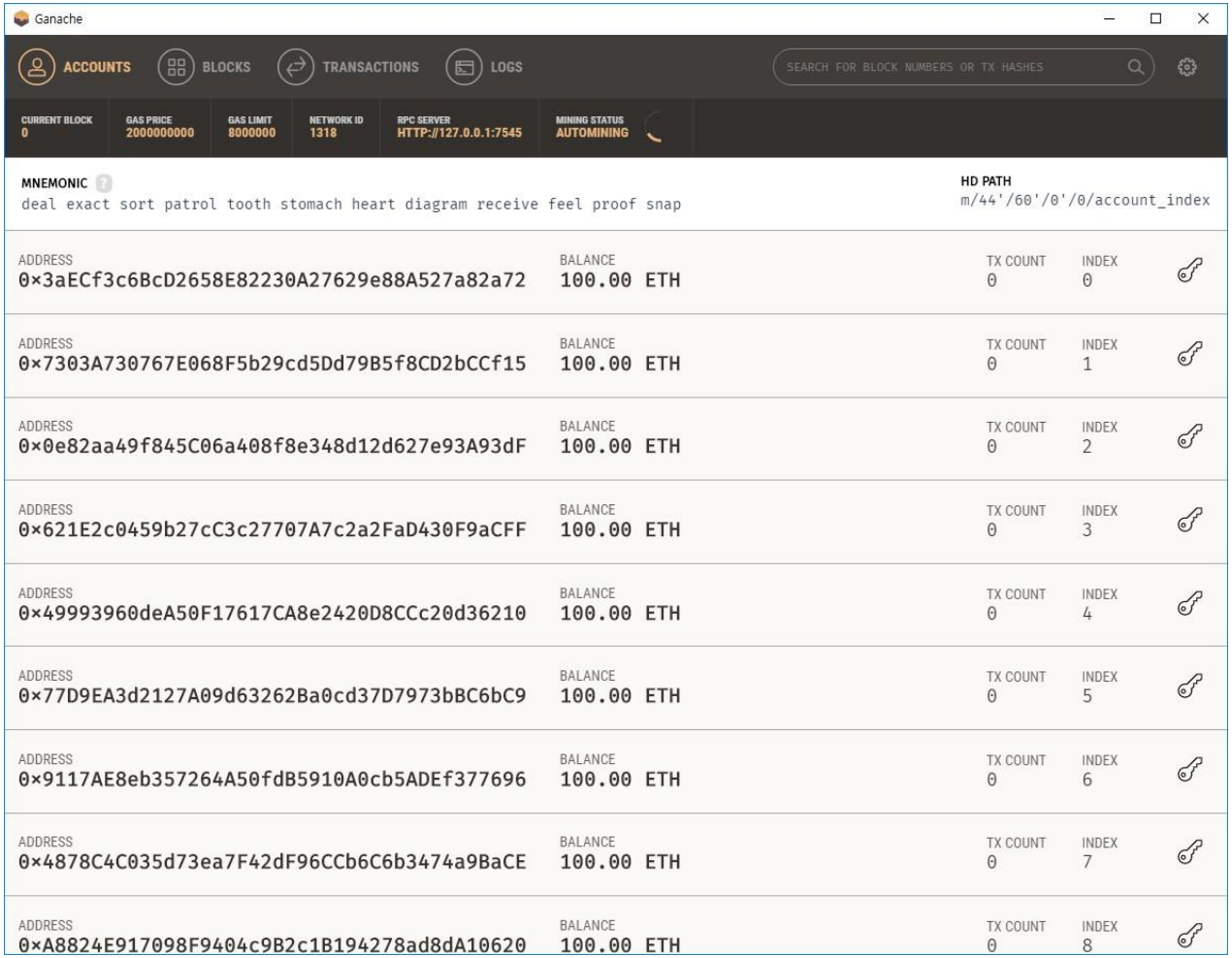
- 가상의 이더리움 네트워크를 생성해서 스마트 컨트랙트를 실행할 수 있도록 해주는 프로그램, 이런 가상 환경을 TestRPC라고 함
- Ganache 다운로드 페이지(http://truffleframework.com/ganache/)에 접속해서 다운로드를 받고 설치



6. 활용예제

6.4 가나슈 실행(2/3)

- 아이콘을 더블 클릭해서 실행을 하면 아래와 같은 화면 출력



6. 활용예제

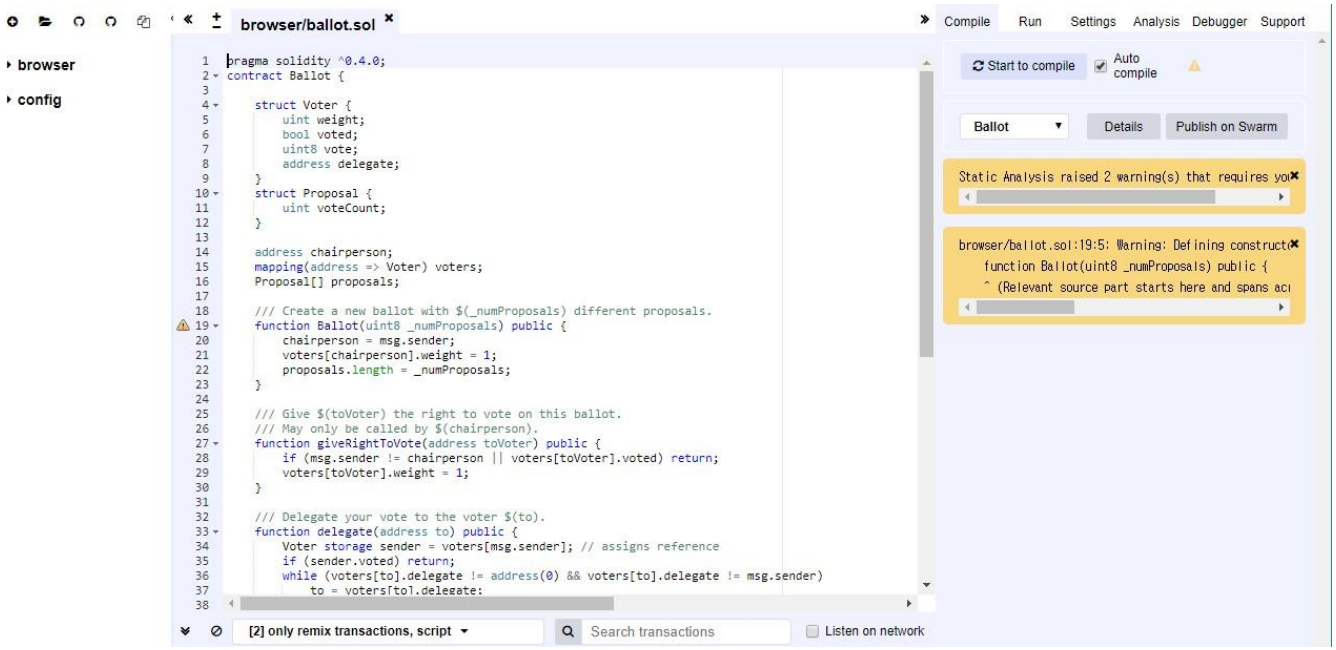
6.4 가나슈 실행(3/3)

- 화면에 대해서 간략히 설명함
- 현재 가상의 이더리움 네트워크가 운영되고 있음
- 해당 네트워크에 접속하려면 http://127.0.0.1:7545로 접속
- 10개의 가상 계정에는 각각 100 이더의 잔액이 충전되어 있음
- 여기까지 하면 이더리움 네트워크는 준비 완료

6. 활용예제

6.5 Remix 접속(1/7)

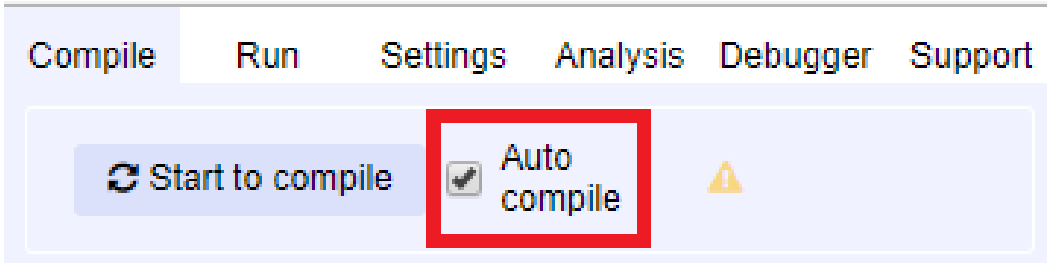
- 웹 브라우저(Chrome 또는 Firefox 추천)를 실행
- 주소창에 https://remix.ethereum.org 를 입력
  - 개발을 위해서 직접 컴파일러를 설치하거나 Ethreum Studio를 설치하는 등 여러 개발 도구들이있으나 Remix는 웹브라우저를 통해서 항상 최신 버전을 사용할 수 있기 때문에 편리함
- 접속이 되면 아래와 같은 기본 화면 출력



6. 활용예제

6.5 Remix 접속(2/7)

- Compile 탭
  - Auto Compile 옵션 체크



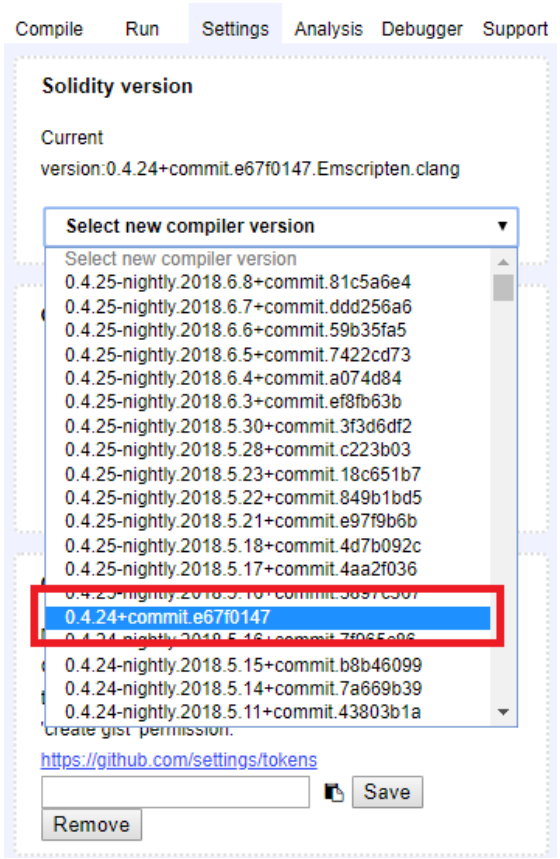
6. 활용예제

6.5 Remix 접속(3/7)

- Settings 탭

  - Solidity 버전을 선택할 수 있는데, 뭔가 안정적으로 보이는 버전 선택

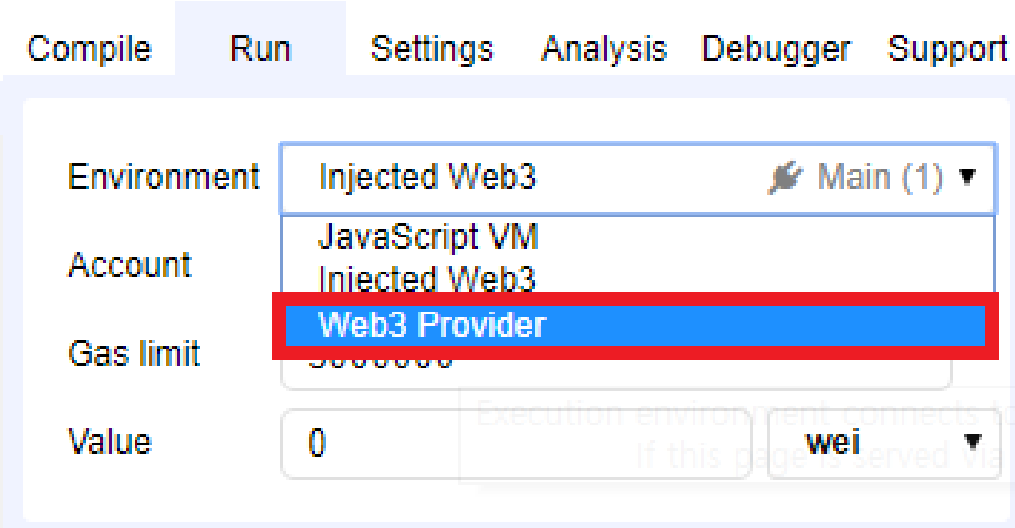
  (아직도 활발하게 개발 중인 언어라서 버전마다 추가/삭제 되는 기능들이 꽤 많이 있음)



6. 활용예제

6.5 Remix 접속(4/7)

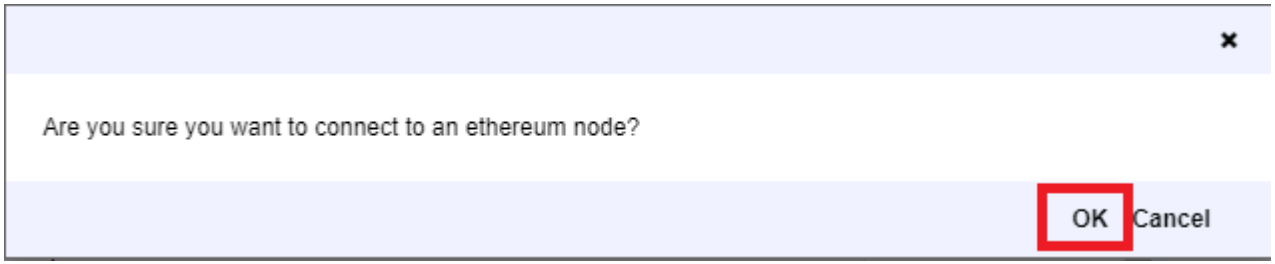
- Run 탭
  - 아까 띄워 놓았던 Ganache와 연결을 해야함
  - Environment에서 Web3 Provider를 선택



6. 활용예제

6.5 Remix 접속(5/7)

- 아래와 같은 확인 창이 뜨는데 OK 클릭



6. 활용예제

6.5 Remix 접속(6/7)

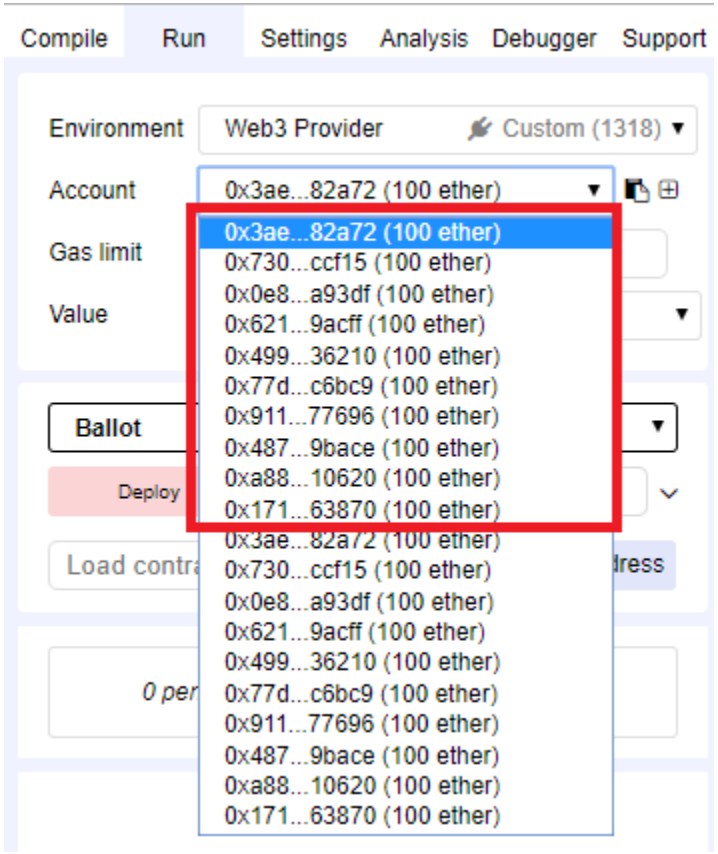
- 주소를 입력하는 창 출력
- Ganache 접속을 위한 주소 http://127.0.0.1:7545 입력

  (기본적으로 localhost는 127.0.0.1로 치환, 만약 접속이 안되시면 127.0.0.1로 직접 입력)

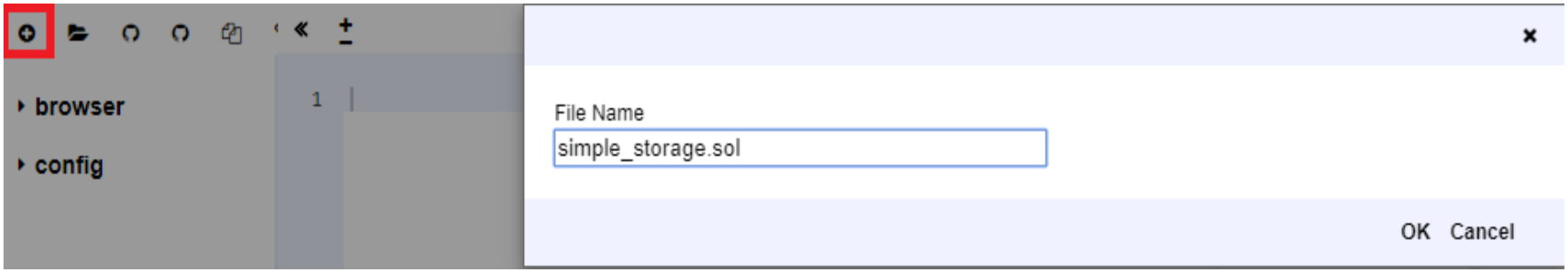
6. 활용예제

6.5 Remix 접속(7/7)

- 정상적으로 접속이 되었다면 Account 부분에 변화가 생김
- Ganache에서 봤던 10개의 계정이 나타나게 됨



- **컨트랙트 작성**
  - 샘플로 열려 있는 ballot.sol 창을 닫고 + 버튼을 눌러서 새 창을 생성, 파일 이름은 simple_storage.sol로 지정하고 OK 클릭



6. 활용예제

6.6 스마트 컨트랙트 예제 실행(2/10)

- 새로 만들어진 창에 아래 코드를 붙여 넣음

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

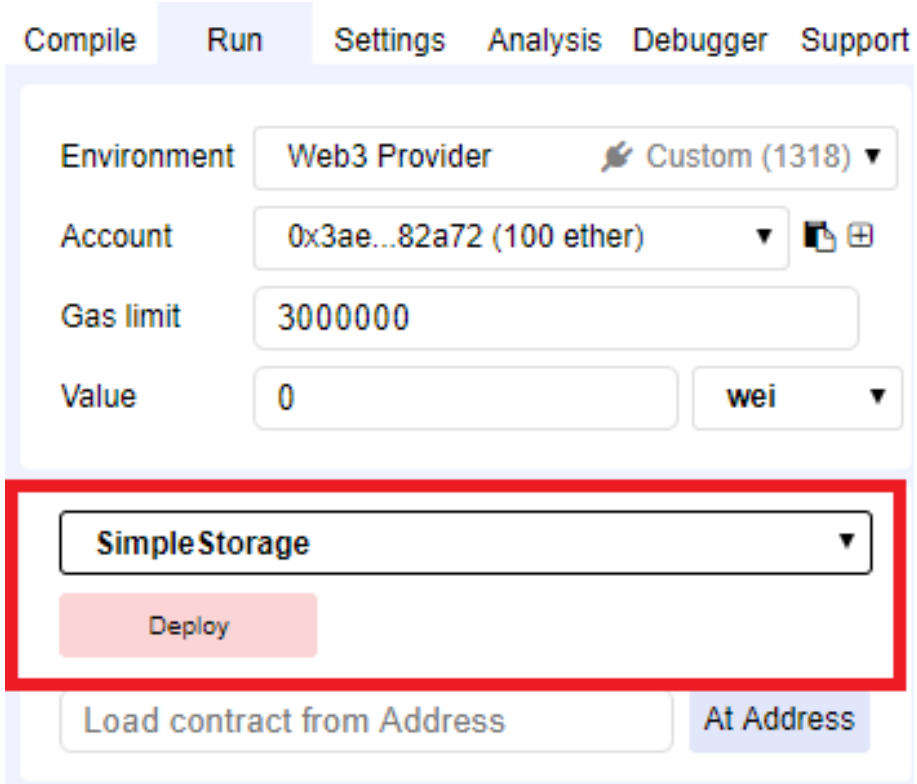
6.6 스마트 컨트랙트 예제 실행(3/10)

- 위의 코드는 스마트 컨트랙트 개발의 Hello World!와 같은 코드
  - solidity 0.4.0 버전을 기준으로 작성
  - SimpleStorage라는 컨트랙트 한 개 포함
  - storedData라는 상태 변수 선언
  - set을 통해서 값 저장
  - get을 통해서 저장된 값 확인

6. 활용예제

6.6 스마트 컨트랙트 예제 실행(4/10)

- **컨트랙트 배포**
  - 코드를 붙여넣기 한 순간 아까 지정한 자동 컴파일 옵션 때문에 자동으로 소스가 컴파일 되었을 것임
  - 컴파일이 완료되면 포함하고 있는 컨트랙트의 이름을 오른쪽 패널에 표시

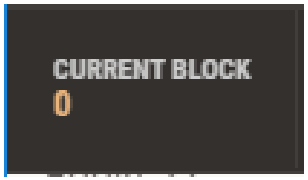


6. 활용예제

6.6 스마트 컨트랙트 예제 실행(5/10)

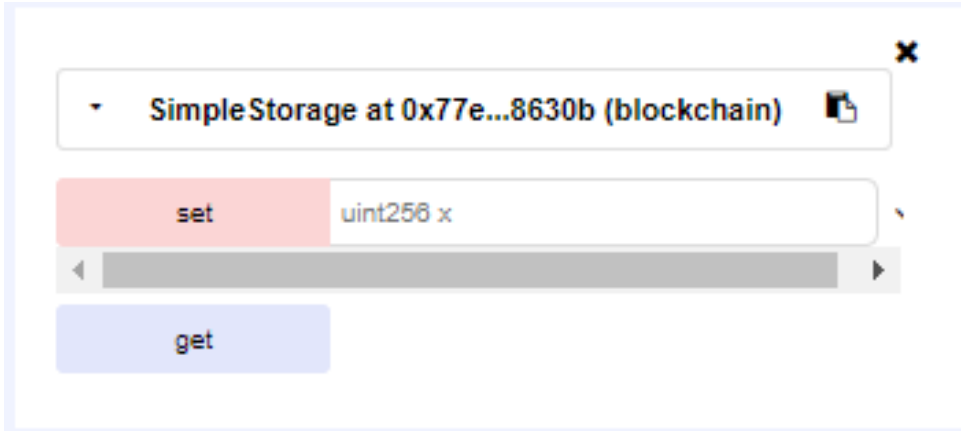
- SimpleStorage 컨트랙트가 컴파일되어 있는 것 확인
- Deploy 버튼을 누르기 전에 Ganache를 다시 한번 살펴봄

  - Current Block에 0이라고 표시된 게 보임



- 이 상태에서 Deploy 버튼을 클릭

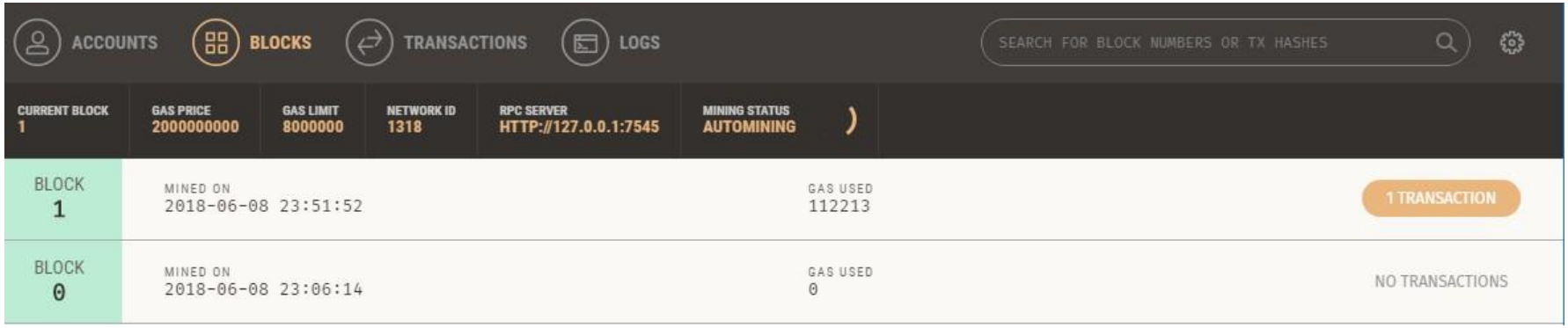
  - Deploy가 되면 오른쪽 패널 아래 새로운 버튼들이 생김



6. 활용예제

6.6 스마트 컨트랙트 예제 실행(6/10)

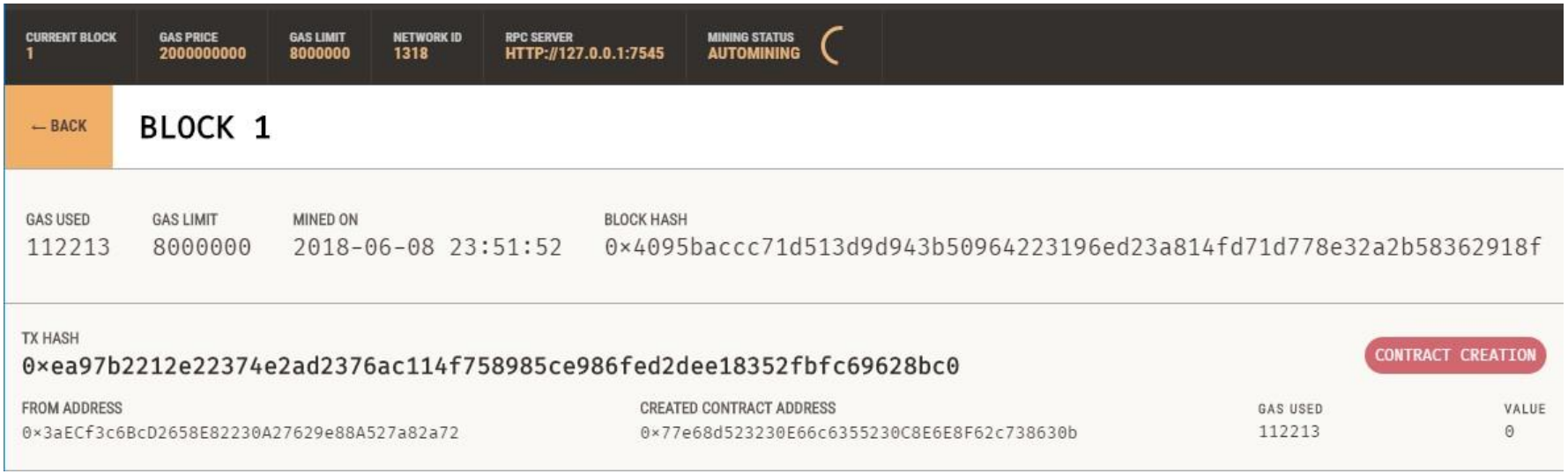
- Ganache의 Current Block를 보면 1로 바뀐 것 확인 가능
  - Deploy와 동시에 transaction을 Ganache 가상 네트워크에 전송했고, 채굴이 일어나서 블록이 1개 생성된 것임
  - Ganche Blocks 탭에서 자세한 내용 확인
  - 블록이 하나 더 생성



6. 활용예제

6.6 스마트 컨트랙트 예제 실행(7/10)

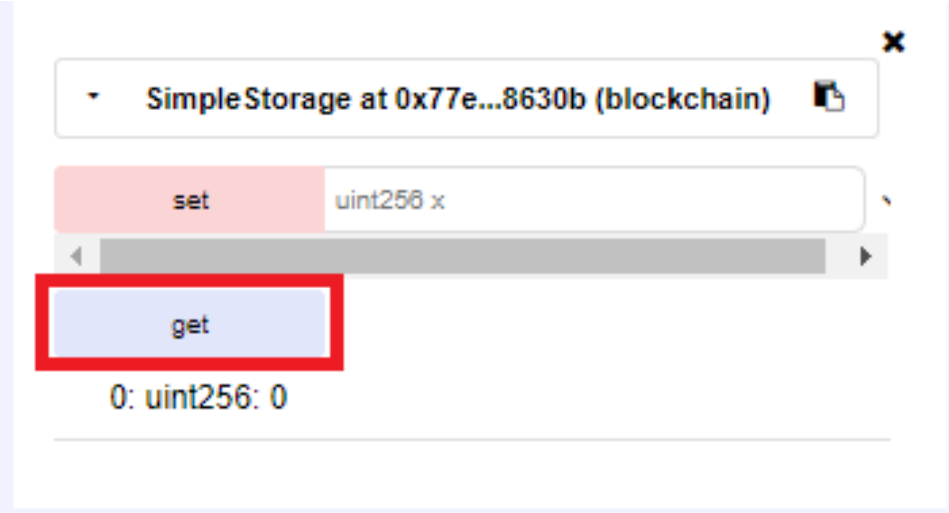
- Transaction을 눌러서 들어가면 방금 Deploy를 하면서 발생시킨 transaction이 들어 있음



- **컨트랙트 실행**

  - set과 get 버튼이 있는데, set은 상태 변수에 값을 저장 "쓰기" 명령이고 get은 상태 변수의 값을 불러오는 "읽기" 명령

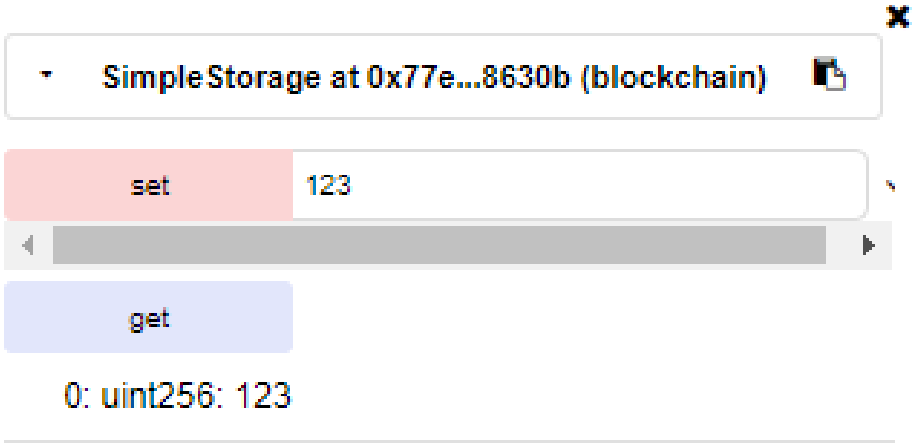
  - 배포 된 상태에서 get 버튼을 눌러서 상태 변수 값 확인



6. 활용예제

6.6 스마트 컨트랙트 예제 실행(9/10)

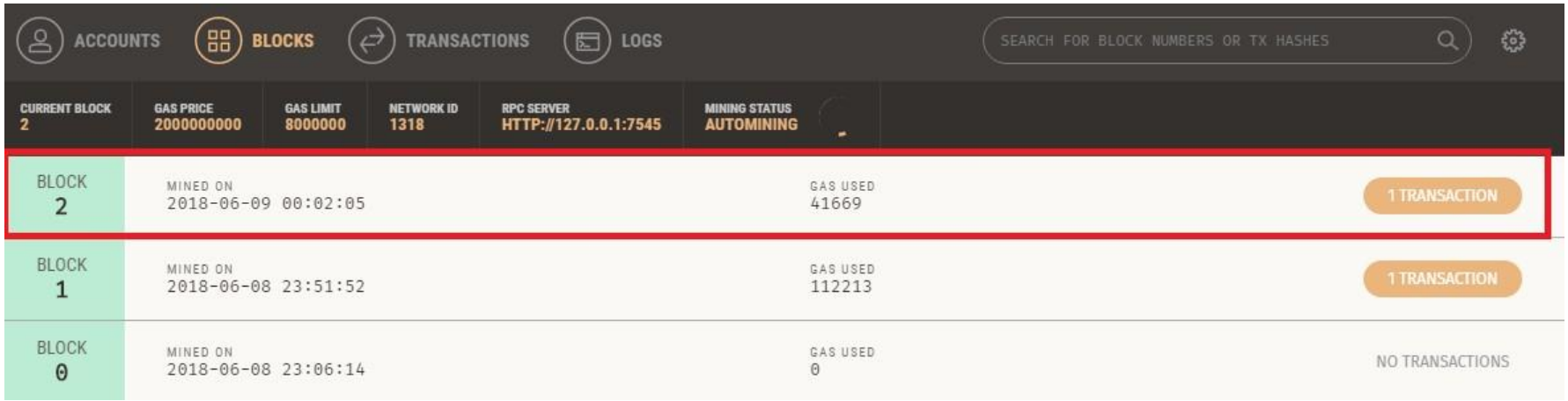
- 상태 변수의 초기 값은 0으로 세팅 되어 있음
  - 그러면 다음으로 set 옆에 입력 간에 원하는 숫자를 입력하고 set은 누른 후 다시 get 클릭



6. 활용예제

6.6 스마트 컨트랙트 예제 실행(10/10)

- 값이 변한 것 확인
  - 지금은 Remix 환경의 함수 실행 버튼을 통해서 컨트랙트의 함수를 실행하지만, web3.js를 이용해서 실제 웹 페이지 상에서 실행할 수 있도록 구현을 하게 되면 그게 바로 DApp이 됨
  - 추가로 Ganache를 한 번 다시 확인해 보시면, 블록이 하나 더 채굴 된 것을 확인



7. FAQ

Q 이더리움은 채굴은 어느정도 시간이 걸리나요?

A 이더리움은 대략 15초에 한 개의 블록이 나오고 있으며, 이 블록에는 많으면 200개 정도의 거래(또는 스마트 컨트랙트)를 수용할 수 있다.

Q Ethereum의 목적은 ?

A 분산 애플리케이션 제작을 위한 대체 프로토콜을 만드는 것이다. 대규모 분산 애플리케이션에 유용할 것이라 생각되는 다른 종류의 제작기법을 제공하며, 빠른 개발 시간, 작고 드물게 사용되는 애플리케이션을 위한 보안, 다른 애플리케이션과의 효율적인 상호작용이 중요한 상황에 특히 주안점을 두고 있다.

8. 용어정리

용어	설 명
이더리움	비탈릭 부테린이 만든 암호화 화폐
스마트컨트랙트	블록체인에 기록된 코드에 의해서 특정 조건이 만족하면 정해진 행동이 자동으로 실행
GPL	GNU General Public License의 약자. 라이선스의 종류 중 하나. 소스의 취득, 수정, 배포, 공개가 자유로우나 최종 배포 시 GPL라이선스를 따라야 함. 수정 배포 시 자체 개발한 소스도 공개해야 함
solidity	이더리움에서 가장 많이 사용되는 스마트 컨트랙트를 위한 자동 언어
이더리움 가상머신	스마트 컨트랙트를 실행하는 장치
분산 애플리케이션	거래나 결제뿐 아니라 계약서, SNS, 이메일, 전자투표 등 다양한 애플리케이션을 투명하게 운영할 수 있게 확장성 제공
DAO	Decentralized Autonomous Organization, 탈중앙화된 자율조직에서 따온 말로 어떤 조직도 소유하지 않고 자율적으로 합의하의 프로세스가 실행되며 분산화된 시스템

Open Source Software Installation & Application Guide