# Chapter 3. 블록체인 개발을 위한 네트워크 구성

네트워크는 개방정도에 따라 (1) 누구나 사용할 수 있는 공중망, (2) 부분적 참여의 허가형 또는 (3) 자신만이 사용하는 사적망으로 나누어 볼 수 있다. 네트워크 망에는 고유번호가 주어져서 식별할 수 있다. 이더리움 사적망을 직접 개설해보게 된다. 망을 사용하려면 필수적으로 계정을 발급해야 하고, 접속하여 클라이언트를 개설하고, 마이닝을 해서 충전을 해본다.

# 1. 네트워크

블록체인 네트워크는 분산으로 작동하여 누구나 참여할 수 있지만, 개방 정도에 따라 종류가 나뉜다.

**공중망(Public Network)**은 인터넷처럼 누구나 참여하고 이용할 수 있도록 개방되어 있다.

반면에 **사적망(Private Network)**은 특정 개인이 개설하여 참여가 제한된 네트워크이다. 개인이 사용한다는 의미에서 사적망이나 사설망이나 호환되는 용어이다.

**허가망(Permissioned Network)**은 관리자 또는 관리 기관이 참여가 허용되는 대상을 정해서 네트워크를 구성한다. 이더리움에서 포크(fork)된 **Quorum** (https://www.goquorum.com/), 리눅스 재단에서 시작한 **하이퍼레저** (https://www.hyperledger.org/), 씨티은행이나 우리나라의 신한, 우리은행 등 금융기관들이 주로 참여하고 있는 **Corda** (https://www.corda.net/), Ripple 등이 해당된다.

|  |  |  |
| --- | --- | --- |
| 개방정도 | 구분 | 설명 |
| 제한 | 사적망 | 어떤 기관이나 개인이 개설하고, 참여하는 권한을 그 기관이나 개인이 결정하여 허락이된 참여자만 읽고 쓸 수 있다. |
| 허락 | 허가망 | 공중망과 사적망 중간쯤에 해당하는 네트워크로서, 참여자를 일부로 제한하여 선정하고, 허락되지 않은 경우에는 참여할 수 없게 구성한다. |
| 무제한 | 공중망 | 어느 누구나 참여할 수 있는 네트워크, 합의뿐만 아니라 읽고 쓰는 권한을 가진다. |

# 2. 공중망

공중망은 개방되어 누구나 사용할 수 있다. 공중망을 사용하는 블록체인의 대표적인 예로 이더리움의 **메인 네트워크**인 홈스테드(Homestead)가 있다.

이더리움의 Ropsten 또는 Rinkeby도 누구나 참여할 수 있는 공중망이지만, 스마트 계약을 개발하고 정식으로 배포하기 전에 시험하는 테스트 용도로 개방되어 있는 **테스트 네트워크**이다.

## 2.1 네트워크 버전 및 ID

### 네트워크/체인(Chain) ID 1번과 3번

이더리움 네트워크에서는 두 개의 식별자(identifier, ID)인 네트워크 ID와 체인 ID를 사용한다.

* 네트워크 ID는 네트워크를 식별하기 위한 고유한 숫자이고,
* 체인 ID는 스마트계약이 속한 블록체인을 식별하기 위한 고유한 숫자. 트랜잭션 내에서 서명 프로세스에서 사용하도록 권장된다. 하드 포크(EIP-155) 이전에는 서명에 네트워크 ID를 사용했지만, 그 이후로는 체인 ID를 사용한다. 특정 체인에서 발생한 트랜잭션을 다른 체인에서 재사용하는 등의 문제를 방지하기 위한 것으로, 서명 검증 시 특정 체인에 대한 ID를 명시적으로 확인할 수 있어서 보안 측면에서 더 안전하다.

네트워크 ID와 체인 ID는 사용 목적이 다르고, 다른 번호를 사용하는 경우가 있을 수 있지만, 특별하게 명시하지 않으면 같은 값으로 사용되는 경우가 대부분이고 두 ID는 동일하다고 보면 된다.

이더리움 공중망은 2015년에 처음으로 시험용 Olympic이 발표되었다. 다음 해 2016년 Frontier로 대체되고, 곧 이어 Homestead로 업그레이드 되었다.

발표하는 버전에는 고유 번호가 붙어 있어 접속할 때 사용한다. 현재 사용되고 있는 **Homestead** 버전이 메인 네트워크로 **1 번**이 붙어 있고, 대응되는 **테스트 네트워크 Ropsten**에는 **3 번**이 고유 번호로 사용되고 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| 네트워크/체인 ID | 명칭 | 구분 | 설명 |
| 0 | Olympic | testnet | 2015년 7월 다음 버전 Morden으로 대체 |
| 1 | Frontier | main | 2016년 3월 **Homestead**로 업그레이드 |
| 2 | Morden | testnet | **Frontier의 테스트 네트워크**. 2016년 종료 |
| 3 | Ropsten | testnet | **Homestead의 테스트 네트워크**. 2016년 발표. 스위치를 --testnet로 주면 접속할 수 있다 (최신 버전에서는 --ropsten으로 그냥 명칭을 적어준다) |
| 4 | Rinkeby | testnet | PoA (Proof of Authority) 방식을 적용하고, --rinkeby 스위치로 접속할 수 있다. |

네트워크를 시작할 때, 네트워크 ID를 반드시 확인해야 한다. ID를 잘못 지정하면 접속되는 네트워크가 달라진다.

더 알아보기: EIP-155

EIP는 Ethereum Improvement Proposal의 약어로 프로토콜의 변경이나 개선을 제안하고 설명하는 표준 문서이다. 제안이 받아들여지면 EIP155와 같이 번호가 부여된다. EIP-155은 서명에 체인 ID를 포함시키기 위한 것으로, 같은 네트워크 ID를 가진 다른 블록체인과의 서명 충돌을 방지하기 위한 목적으로 도입되었다. 해커가 중간에 보내는 데이터를 가로채어 다시 보내는 것을 재전송 공격(replay attack)이라고 하는데, 이를 막기 위해 ChainID를 적어 놓으면, 해커가 자신의 ChainID로는 재전송을 할 수 없게 된다.

EIP는 일반적으로 다음과 같은 세 가지 유형으로 구분할 수 있다.

1. **코어(EIP-1 ~ EIP-999):** 이더리움 프로토콜의 핵심 부분을 변경하는 제안
2. **ERC (EIP-20, EIP-721, EIP-1155 등):** ERC는 Ethereum Request for Comments의 약자로, 이더리움 스마트 계약의 표준을 정의하는 제안. 나중에 배우는 NFT, DeFi 등이 ERC 표준에 따라 개발되면 상호 운용성이 당연히 향상된다.
3. **META (EIP-1001, EIP-1011 등):** 프로세스나 표준에 대한 변경을 다루는 제안

### 자신의 사적망 체인 ID는 어떤 번호를 사용할까?

사적망(Private Network)을 설정할 때 사용되는 네트워크 ID는 사용자가 임의로 1부터 65,535까지의 정수 값으로 설정할 수 있지만, 충돌을 피하 위해 노력해야 한다.

우선 이더리움의 메인, 테스트망에서 공식적으로 점유하는 4번까지는 피하고 그 이후의 번호를 사용하자.

또한 널리 사용되고 있는 ID와 충돌이 나지 않도록 하는 편이 좋다. 이더리움 포크가 일어나면서, 이더리움 클래식은 체인 ID로 61번을, 테스트망은 62번을 각 각 사용한다.

이미 많은 번호가 점유되어 사용되고 있으므로, 본인이 사용하려고 하는 번호가 이미 사용되고 있는지 확인하고 주의한다 (https://chainlist.org/ 에서 번호를 확인할 수 있다).

## 2.2 메인 네트워크

### 서버가 필요없다

보통은 애플리케이션을 배포하려면 서버가 있어야 한다. 자신이 운용하는 서버나 빌려서 사용하는 호스트 서버가 필요하다.

공중망을 사용한다면 어떨까?

공중망에서 스마트 계약을 실행한다면, 이를 위한 서버를 별도로 장만할 필요가 없다. 공중망이 서버 호스트 역할을 한다고 볼 수 있다.

공중망의 블록체인은 발생하는 모든 거래가 기록되고 누구에게나 공유되는 환경이다. 자신이 구현한 어플리케이션도 공중망에서 실행하고 운영할 수 있다.

### 화폐은 어떻게 획득할까?

화폐를 획득하려면, 채굴을 해야 한다. 채굴 즉 마이닝을 하게 되면 이더(Ether)를 획득할 수도 있다.

왜 "획득할 수도"라고 했을까? 그 이유는 마이닝해서 Ether를 획득하는 것이 극히 어렵기 때문이다. 비트코인이나 이더리움이나 암호화폐를 획득하려면 공중망에 연결하여 마이닝에 참여해야 한다. 그러나 마이닝을 통해 암호화폐를 얻을 가능성은 거의 없다고 봐도 좋다.

마이닝이 아니라면, 직접 암호화폐를 구매하거나 와이어(Wyre)와 같은 블록체인 기반 국제 송금 업체를 통해 구매할 수 있다.

예를 들어 MetaMask에서는 공중망에 쓰이는 Ether를 체크카드를 통해 구매하는 서비스를 제공한다. 다른 사적망이나 테스트망과 달리 **Ether가 유통**되기 때문에 거래를 테스트해서는 안되며 주의를 기울여야 한다.

오늘 가격이 올랐다 또는 떨어졌다하고 뉴스에 나오는 경우, 메인네트워크의 Ether를 말하는 것이다.

### 화폐가 없으면?

공중망에는 암호화폐인 Ether가 없다면, 참여는 할 수 없는 것일까?

그렇지는 않다. 가스비가 필요한 경우는 당연히 참여가 불가능하지만, 그렇지 않은 경우, 정보를 조회하거나 마이닝에 참여하거나 하는 등의 가스비가 없어도 되는 작업에는 당연히 참여할 수 있다.

### 운영 상황

Ethernodes.org에서는 메인 네트워크가 운영되고 있는 상황을 확인할 수 있다 (https://www.ethernodes.org/).

* 1번 메인 네트워크는 미국에서 과반 가까이 쓰이고 있고, 다음이 독일이다. 그 뒤를 프랑스, 싱가포르, 영국, 일본이 따르고 있지만 순위는 자주 바뀐다.
* 클라이언트는 리눅스 운영체제가 압도적이고, geth가 많이 활용되고 있다.

메인 네트워크에서 발생하는 거래관련 정보는:

* Ethstats.net https://ethstats.net/ 에서 거래 및 블록 관련 통계를 볼 수 있다.
* Etherscan.io https://etherscan.io 에서도 비슷하게 실시간 발생하는 거래를 조회할 수 있다.

### 접속 및 동기화

메인 네트워크에 접속해 보자. 그렇게 되면 자신의 CPU, 메모리 자원이 할당되어 사용되게 되고, 이로 인해 컴퓨터가 느려질 수 있다. 블록체인의 동기화와 마이닝에 컴퓨터의 자원이 소모될 수 있다.

메인 네트워크에 연결해서 블록체인에 있는 블록 정보를 내려받아 자신의 컴퓨터에 저장하는 것을 동기화(synchronization)라고 한다. 메인 네트워크의 블록체인 규모가 커서 동기화하려면 시간이 꽤 걸릴 수 있다. 그래서 전체 또는 일부만 동기화시키는 방법을 지정할 수 있다. 동기화 방식은 스위치 --syncmode로 full, fast, snap, light 방식 중 선택해서 지정할 수 있다.

* full 옵션(full sync): 블록체인에서 가장 첫 번째로 생성되는 genesis block부터 시작해서 현재까지 모든 거래를 내려받고 검증하는 것을 full sync라고 한다. 모든 거래만 내려받는 것이 동기화라고 오해해서는 안된다. 거래를 검증하려면 계정의 상태도 확인해야 하는데, 따라서 계정정보도 모두 내려 받아야 한다 (이런 정보는 state trie에 저장되어 있다).
* fast 옵션(fast sync): 최근 블록의 상태로부터 블록을 동기화하는 방식으로 작동하는데, 모든 거래를 내려받고 검증하는 것이 아니라, Transaction Receipts과 State Trie를 활용하여 상태를 동기화한다. 여기서 상태(State)는 모든 계정의 상태를 포함하는 데이터 구조로서, 계정의 잔액, 코드, 저장소 등을 나타낸다. 모든 거래를 내려받고 검증하는 것이 아니라 빠르다. 1.10버전부터는 fast sync를 지원하지 않는다.
* snap 옵션(snap sync): fast sync의 대기, 입출력시간을 줄이기 위해 개선한 방법이다. 1.10버전에는 snap sync를 선택하면, 더 빠르게 동기화할 수 있어 많이 쓰인다. 일정 블록 간격마다 "state snapshot"을 사용하여 상태를 동기화하는 옵션이다.
* light 옵션(light sync): 버전 1.5.2부터 제공되는 **light node**로 모든 블록체인을 저장하지 않고, 블록헤더만 동기화할 수 있다. State trie도 내려받지 않는다. 필요한 경우, 피어에게 계정상태 등을 의존해야 한다. 따라서 피어가 없으면 문제가 될 수 있다. 용량이 제한되어 있는 모바일 폰 등에서 사용하면 적합하다.

이 중에서 **full sync**를 선택하면, 동기화에 CPU와 메모리를 상당히 소모해서 컴퓨터를 사용하기 매우 불편한 정도가 될 수 있다. 모든 거래를 처음부터 동기화하기 때문이다. 매일 백 만 건이 넘는 거래가 발생하고, 현재까지의 거래 건수는 10억 건이 넘는다. 이 모든 거래 데이터를 동기화하려면 성능이 좋은 컴퓨터도 10 일은 걸린다고 한다.

최근 1.8버전에서는 많이 개선되었다는 반응이 있기는 하다. 하지만 메인 네트워크는 데이터의 크기가 130 GB를 넘고, 테스트 네트워크는 그보다 적지만 수십 GB를 내려받아야 한다.

또한 네트워크를 빠르게 동기화 하려면 --syncmode snap을 선택할 수 있다 (버전 1.10 이전에서는 --syncmode fast). 그러면 블록체인 전체를 다운로드 하지 않는다.

또한 메모리까지 --cache=1024로 설정할 수 있다. 메모리 cache는 16MB단위로 증가할 수 있고, RAM크기에 따라 설정할 수 있다. 1024는 약 1G 정도를 배정한 경우이다.

geth --syncmode snap --cache=1024 console

따로 --datadir 스위치를 명시하지 않으면, 아래 디렉토리가 생성되고 keystore, chaindata 등의 관련 데이터가 저장된다. 생성되는 디렉토리는 운영체제 별로 다르다.

* 리눅스: ~/.ethereum (blockchain, keystore), ~/.ethash (DAG)
* 맥 OS X: ~/Library/Ethereum, ~/Library/Ethereum
* 윈도우는 디렉토리가 다를 수 있다. C:/Users/<사용자명>/AppData/Roaming/Ethereum

geth 명령에 console을 붙이면 동일한 단말(터미널 또는 cmd 윈도우)의 화면에 로그(log) 내용을 출력한다. 이렇게 하나의 단말을 사용하면, 명령어를 입력하는 순간에도 로그가 계속 출력되어 번거롭다.

두 개의 단말을 사용하면 편리하다. 한 쪽은 서버를 (분산시스템은 서버가 없으니까 정확한 단어는 아니다). 다른 쪽은 클라이언트를 실행한다는 것이다. 로그가 여러 줄에 걸쳐 나오더라도 서버 창에서 출력되므로, 클라이언트는 방해 받지 않고 작업을 할 수 있게 된다.

## 2.3 테스트 네트워크

공중망 메인네트워크가 있는데 왜 테스트망이 필요할까?

테스트 네트워크는 메인의 복사본 쯤으로 볼 수 있다. 테스트 네트워크는 말 그대로 어플리케이션을 테스트하는 네트워크이다.

실제 Ether가 통용되는 상용 어플리케이션을 메인 네트워크에 바로 올릴 수는 없지 않은가? 메인 네트워크에 배포하기 전 잘 작동되는지 테스트를 해보는 네트워크가 필요하다.

현재 운영되는 **메인 네크워크 Homestead**에 대응되는 테스트 네트워크는 **Ropsten**이다. 앞서 보듯이, 현재 사용할 수 있는 테스트 네트워크는 Ropsten 외에도 Kovan, Rinkeby이 있다.

### 접속 및 동기화

테스트 네트워크를 사용하는 방법은 geth 버전에 따라 차이가 있다. 다음 코드에서는 --testnet 옵션을 입력해서 테스트 네트워크를 사용한다. 최신 버전에서는 geth --ropsten이라고 해야 한다. geth -h는 geth 프로그램을 실행시키면서 사용할 수 있는 옵션들의 도움말을 보인다. 어떤 형태로 써야 하는지 -h 옵션을 이용해서 확인해도 된다.

별도로 명시하지 않으면 데이터베이스는 사용자 디렉토리 아래 .ethereum/testnet/에 만들어 진다.

pjt\_dir> geth --testnet (버전 1.10 이후 ```geth --ropsten```)  
  
INFO [03-17|12:39:05.806] Maximum peer count ETH=25 LES=0 total=25  
INFO [03-17|12:39:05.807] Starting peer-to-peer node instance=Geth/v1.8.23-stable-c9427004/linux-amd64/go1.10.4  
INFO [03-17|12:39:05.807] Allocated cache and file handles database=\*\*/home/jsl/.ethereum/testnet/\*\*geth/chaindata cache=512 handles=524288  
INFO [03-17|12:39:06.071] Persisted trie from memory database nodes=355 size=51.89kB time=780.85µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B  
INFO [03-17|12:39:06.072] Initialised chain configuration config="{ChainID: 3 Homestead: 0 DAO: <nil> DAOSupport: true EIP150: 0 EIP155: 10 EIP158: 10 Byzantium: 1700000 Constantinople: 4230000 ConstantinopleFix: 4939394 Engine: ethash}"  
INFO [03-17|12:39:06.072] Disk storage enabled for ethash caches dir=/home/jsl/.ethereum/testnet/geth/ethash count=3  
INFO [03-17|12:39:06.072] Disk storage enabled for ethash DAGs dir=/home/jsl/.ethash count=2  
INFO [03-17|12:39:06.072] Initialising Ethereum protocol versions="[63 62]" network=3  
INFO [03-17|12:39:07.179] Loaded most recent local header number=1152 hash=4ec42f…556803 td=1069386844 age=2y4mo6d  
INFO [03-17|12:39:07.179] Loaded most recent local full block number=0 hash=419410…ca4a2d td=1048576 age=49y11mo2d  
INFO [03-17|12:39:07.179] Loaded most recent local fast block number=230 hash=4057f4…3be8bb td=195973576 age=2y4mo6d  
INFO [03-17|12:39:07.179] Loaded local transaction journal transactions=0 dropped=0  
INFO [03-17|12:39:07.179] Regenerated local transaction journal transactions=0 accounts=0  
INFO [03-17|12:39:10.743] New local node record seq=2 id=150304bb4b4a1a90 ip=127.0.0.1 udp=30303 tcp=30303  
INFO [03-17|12:39:10.744] Started P2P networking self=enode://3166b07b53259f5bb77d46806375cb0164d68ea48b48e0ea75ea34fbe07d0b3486f28c1bde74fe7892a5e1e513a6694aa295b19d08ed2310d8d0f301b432b85d@127.0.0.1:30303  
INFO [03-17|12:39:10.745] IPC endpoint opened url=/home/jsl/.ethereum/testnet/geth.ipc  
INFO [03-17|12:39:20.744] Block synchronisation started

위 로그 줄 5의 [INFO]를 보면, cache에 512가 할당되어 있다. 블록체인의 동기화를 좀 빠르게 하려면 다음과 같이 --cache를 늘려 잡을 수 있다. 이는 앞의 메인 네트워크에서 설정하는 방식과 동일하다.

geth --ropsten --syncmode snap --cache=1024 console

위 명령어를 실행하여 테스트 네트워크에 접속할 수 있다. 맨 뒤 console 명령어는 콘솔 프롬트프 >를 출력한다. 그 프롬프트에 아래에 보인 것처럼 admin.nodeinfo를 입력하고 출력되는 내용을 확인해본다. 네트워크와 체인 ID 모두 3번, 그러니까 Ropsten으로 올바르게 설정되어있다.

geth> admin.nodeInfo  
{  
 enode: "enode://3166b...생략...2b85d@127.0.0.1:30303",  
 enr: "0xf896b...생략...2765f",  
 id: "150304bb4b4a1a906d3b92a96db6c1cda9b0f7f01618b5a7ed38d246859ff174",  
 ip: "127.0.0.1",  
 listenAddr: "[::]:30303",  
 name: "Geth/v1.8.23-stable-c9427004/linux-amd64/go1.10.4",  
 ports: {  
 discovery: 30303,  
 listener: 30303  
 },  
 protocols: {  
 eth: {  
 config: {  
 byzantiumBlock: 1700000,  
 chainId: 3,  
 constantinopleBlock: 4230000,  
 daoForkSupport: true,  
 eip150Block: 0,  
 eip150Hash: "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",  
 eip155Block: 10,  
 eip158Block: 10,  
 ethash: {},  
 homesteadBlock: 0,  
 petersburgBlock: 4939394  
 },  
 difficulty: 1048576,  
 genesis: "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",  
 head: "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",  
 network: 3  
 }  
 }  
}

### 충전

테스트망에서 프로그램을 구현하고 실행시켜보려면 Ether가 필요하다. 그러니 실제 화폐 Ether가 있어야 하는 것은 아니다. 테스트망에서 사용하는 '가짜' Ether만 있으면 된다. '가짜'라는 단어는 어떤 속임수가 있다는 의미는 아니고, 가치가 없다는 정도로 이해하자.

왜 공중망 또는 테스트망에 Ether가 필요할까? 블록을 생성하는 거래에는 수수료를 지불해야 한다. 그 수수료가 채굴한 사람에게 지불이 된다.

그렇다면 테스트망에서 충전은 어떻게 하는가? testnet에 사용할 수 있는 Ether는 (1) 마이닝하거나, (2) faucet에서 충전할 수 있다.

faucet에서 Robsten 계정을 충전하는 것은 어렵지 않다. 말 그대로 수도꼭지를 틀어달라고 요청하면 된다. https://faucet.ropsten.be/ 에 접속하여 자신의 계정으로 충전을 요청할 수 있다. 트래픽 상황에 따라 시간이 좀 걸릴 수 있으니 바로 충전이 되지 않는다고 조급해 하지 말자.

마이닝으로 충전을 할 경우에는 먼저 계정을 만들어야 하는데, 마이닝 보상의 입금계정이 필요하기 때문이다.

geth --testnet account new

마이닝이라고 하면 좀 어렵게 들릴 수 있다. 실물경제에서 화폐는 은행에서 인출해서 얻을 수 있는데, 암호화폐는 그렇지 않고 마이닝을 통해서 획득할 수 있다. 마이닝을 통해서 발생한 보상이 저장되는 통장과 같은 계정이 필요하다. 따라서 위 명령어로 계정이 만들어 놓지 않으면 마이닝은 가능하지 않다.

1시간 정도 마이닝하면 10 ether는 할 수 있다. 아침 6시 ~ 오후 1시까지 마이닝을 해서 60 ether 정도를 충전할 수 있었다.

geth --testnet --rpc --mine console

# 3. 사적망

사적망은 **개인이 개설**해서 사용하는 네트워크로 개방되어 있지 않고, 다른 사람들이 참여하는 것이 어렵다. 자신이 사용하려고 만들었기 때문에 **분산원장으로 본래의 목적인 공유를 할 수 없다**.

사적망은 완성된 애플리케이션을 공중망에 배포하기 전, 개발하기 위한 용도로 활용한다. 역시 Ether를 사용하지만 실제 가치가 없을 수 밖에 없다. 개인이 자유롭게 개설해서 사용하고 비용이 들지 않는다. 자신이 거래를 발생하고, 또한 마이닝까지 수행하게 되는 것이 보통이다.

## 3.1 Dev Mode

사적망을 설정없이 쉽게 개설할 수 있을까?

매우 간편하게 사적망을 개설하는 방법은 개발 옵션 --dev 만을 주면 된다. 임의의 ChainID (아래 로그에 1337이 출력되고 있다), 계정 및 기본 설정이 주어진다. --maxpeers 0 으로 자동 설정되어 개발자 혼자 사용하게 된다.

pjt\_dir> geth --dev  
  
INFO [03-17|16:05:41.295] Maximum peer count ETH=25 LES=0 total=25  
INFO [03-17|16:05:44.857] Using developer account address=0xFaB10f1929411457aA821A620A406cB1Cd96Ad43  
INFO [03-17|16:05:46.301] Starting peer-to-peer node instance=Geth/v1.8.23-stable-c9427004/linux-amd64/go1.10.4  
INFO [03-17|16:05:46.420] Writing custom genesis block  
INFO [03-17|16:05:46.578] Persisted trie from memory database nodes=12 size=1.79kB time=54.971µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B  
INFO [03-17|16:05:46.613] Initialised chain configuration config="{ChainID: 1337 Homestead: 0 DAO: <nil> DAOSupport: false EIP150: 0 EIP155: 0 EIP158: 0 Byzantium: 0 Constantinople: 0 ConstantinopleFix: 0 Engine: clique}"  
INFO [03-17|16:05:46.653] Initialising Ethereum protocol versions="[63 62]" network=1337  
...생략...

geth를 실행시키면, 위와 같이 어떤 작업이 진행되고, 어떤 상태인지 지속적으로 로그가 화면에 출력되고 있다. 블록체인과 상호작용하면서 처리 과정을 확인하는 등 필요한 경우 참조하면 된다.

## 3.2 자신의 사적망 구성

자신의 사적망을 개설하여 구성해 보자. 앞서 설명한 공중망은 시제품을 만들면서 테스트하기 위해서는 실제 화폐가 필요하기 때문에 낭비일 수 밖에 없다. 반면에 사적망은 공중망과 비교해서 환경에 차이가 없어 테스트하기에 딱 좋다.

여기서는 ChainID, 데이터디렉토리 등의 설정을 주어서 사적망을 만들게 된다. 특히 스위치로 --datadir를 설정하여 데이터 디렉토리를 지정해야 한다. 이를 별도로 지정하지 않으면 메모리에서 구동되다가, 저장되지 않은채 사라지게 된다.

### 단계 1: genesis.json 설정

사설망을 구성하려면, genesis 설정으로 블록체인을 초기화해야 한다. genesis 블록은 블록 체인에서 처음으로 생성되는 블록이고, genesis.json 파일을 이용해서 설정된다.

genesis.json 파일의 구성은 스스로 해도 되지만, 보통 이더리움에서 제공된 genesis.json 예제를 사용하면 편리하다 (참조: https://geth.ethereum.org/docs/interface/private-network)

다음 표는 genesis.json 파일에서 사용되는 설정들을 설명한다.

|  |  |
| --- | --- |
| 설정 | 설명 |
| config | 설정하지 않아도 된다. |
| mixhash | 256 bit hash 값. 블록 헤더의 nonce, timestamp, 앞 블록의 hash, 마이너 주소를 합쳐서 (mix) 만들어진 hash. 블록에 대해 충분한 작업 증명이 되었다는 의미로 쓰임. |
| nonce | 64 bit hash 값. 이 값과 블록체인의 해시값이 더해지고, 이 값을 맞추려면 일정한 양의 수학적 계산이 필요하다. 따라서 블록이 PoW에 따라 마이닝되고 검증되었다는 의미이다. |
| difficulty | 마이닝 난이도로서, 당연히 높으면 마이닝하려면 오래 걸림. 블록에서는 이전 블록의 1) 난이도와 2) timestamp에 따라 결정되지만 genesis block은 이전 블록이 없으므로 임의로 정하게 된다. 낮게 설정해서 대기시간을 줄이는 편이 좋다. |
| alloc | 사전 판매된 계정과 잔고. "alloc": {"0x..": { "balance": "Wei" }} 로컬은 충전이 쉬워 비워놓음. |
| coinbase | 160 bit 주소. 마이닝 보상을 가져가는 계정 주소. 실제 **마이닝하면서 정해지기 때문에 어떤 주소**를 적어도 좋다. |
| timestamp | 블럭 사이의 timestamp길이가 짧으면 난이도를 높이게 된다. |
| parentHash | Keccak 256-bit hash로 부모 블록 헤더 (nonce와 mixhash포함) |
| extraData | 최대 32 byte. 다른 용도를 대비한 공간. |
| gasLimit | 블록 당 최대 소비 제한 gas. 높게 설정해서 out-of-gas과 같은 오류를 줄일 수 있다. |

또한 config에 다음 옵션을 설정할 수 있다.

* ChainId - **네트워크 ID와 같은 값**으로 정해서 적는다. ID번호 4 이하는 공중망에 배정되었으므로 사용하지 않는다. **사설망의 ChainId가 동일하면 Peer로 연결**될 수 있다. 즉 **공중망의 동일한 ChainId를 사용하는 다른 컴퓨터**가 연결될 수 있다. 이 경우 bootnodes를 정해서 연결될 Peer를 제한할 수 있다.
* homesteadBlock - 1번 main 네트워크에 접속을 하면 homesteadBlock이 분기된 1150000, Morden은 494000로 설정되어 있다. 그 블록에서 각 각 hard fork가 일어났기 때문이다. 0으로 설정하면, 첫 Genesis 블록부터 Homestead라는 의미이다.
* eip150Block - EIP-150은 가스 가격(Gas Price) 계산 방식을 개선하기 위한 제안이다. eip150block은 EIP-150의 일부로, 이 개선 제안이 어떤 블록부터 적용되기 시작하는지를 나타내는 값이다. 보통은 하드 포크가 발생하는 특정 블록 번호로 설정한다. 0으로 설정되어 있다면, EIP-150의 변경 사항이 Genesis 블록부터 적용된다는 것을 의미하고, 즉 Ethereum 블록 체인이 시작될 때부터 새로운 가스 가격 계산 방식이 적용되었음을 나타낸다.
* eip155Block - EIP155는 서명에 체인 ID를 포함시켜서 같은 네트워크 ID를 가진 다른 블록체인과의 서명 충돌을 방지하기 위한 목적으로 사용한다. eip155block이 0인 경우, Genesis 블록부터 서명에 체인 ID가 포함되어 있었다는 뜻으로 특별한 조치가 필요하지 않다는 의미이다.
* eip158Block - EIP158은 상태 트리(State Trie) 구조를 변경하는 제안으로 공계정과 관련하여, 이를 없는 것으로 처리하여 블록체인을 절약하기 위한 것이다.

다음은 우리가 적용하게 될 \_genesis.json파일이다. 파일 이름에 우리가 만들었다는 것을 구별하기 위해 맨 앞 언더스코어 (\_)를 일부러 붙었다.

JSON 파일에는 주석문을 붙일 수 없지만, 여기서는 편의상 설명을 붙이기 위해 '#' 기호를 삽입하였다. 실제 파일에 저장할 때, '#'이후의 문자열은 모두 삭제한다.

예를 들어, "chainId": 412, # 5 이상의 숫자만 사용 이 문장은 # 이후의 주석을 제거하고 ---> "chainId": 412, 이렇게 저장하자.

[파일명: "\_genesis.json"]  
{  
 "config": {  
 "chainId": 412, # 5 이상의 충돌나지 않는 임의의 수를 사용하는 편이 좋다.  
 "homesteadBlock": 0,  
 "eip150Block": 0,  
 "eip155Block": 0,  
 "eip158Block": 0,  
 "byzantiumBlock": 0, # constantinopleBlock설정하면서 같이 설정  
 "constantinopleBlock": 0, # 2019년 2월 EIP 145에 따른 업그레이드. 이때 byzantiumBlock도 같이 설정  
 "petersburgBlock": 0 # St. Petersburg 업그레이드 (EIP 1283)는 Constantinople과 같은 블록에서 적용  
 },  
 "nonce": "0x0000000000000033",  
 "timestamp": "0x0",  
 "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",  
 "gasLimit": "0x8000000", # 적게 설정하면 out-of-gas 오류 발생  
 "difficulty": "0x100", # 0x10이나 더 낮게 0x0으로 해도 된다.  
 "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",  
 "coinbase": "0x3333333333333333333333333333333333333333",  
 "alloc": {}  
}

### 단계 2: geth 네트워크 초기화

init 명령어를 사용해서 방금 작성한 \_genesis.json을 읽어 블록체인을 설정한다.

앞의 **main, testnet은 공중망이고, 따라서 이미 블록체인이 이미 구동 중이기 때문에** 별도의 설정이 당연히 불필요하다. 반면에 사설망은 스스로 네트워크를 생성해야 할 뿐만 아니라, **블록체인을 설정, 생성**하는 것이 필요하다.

다음과 같이 --datadir을 프로젝트 디렉토리 아래 eth에 설정하기로 하자. 디렉토리는 꼭 이 명칭으로 하지 않고 임의로 정해도 된다. 그리고 명령어 뒤에서 볼 수 있듯이 init 명령으로 genesis.json을 읽어서 사설망을 초기화하여 생성하게 된다.

pjt\_dir> geth --datadir .\eth init \_genesis.json

다음은 윈도우에서 실행한 결과를 보인다. 로그를 읽어보면, 현재 디렉토리에 올바르게 설정되고 있다는 것과 마지막 줄에 성공적으로 생성이 되었다는 메시지 Successfully wrote genesis state를 출력하고 있다.

INFO [09-16|23:52:23.562] Maximum peer count ETH=50 LES=0 total=50  
INFO [09-16|23:52:23.576] Set global gas cap cap=50,000,000  
INFO [09-16|23:52:23.584] Allocated cache and file handles database=C:\\Users\\jsl\\Code\\201711111\\eth\\geth\\chaindata cache=16.00MiB handles=16  
INFO [09-16|23:52:23.630] Opened ancient database database=C:C:\\Users\\jsl\\Code\\201711111\\eth\\geth\\chaindata\\ancient/chain readonly=false  
INFO [09-16|23:52:23.641] Writing custom genesis block  
INFO [09-16|23:52:23.647] Persisted trie from memory database nodes=0 size=0.00B time=0s gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B  
INFO [09-16|23:52:23.663] Successfully wrote genesis state database=chaindata hash=5704d0..9bc5b0  
INFO [09-16|23:52:23.673] Allocated cache and file handles database=C:\\Users\\jsl\\Code\\201711111\\eth\\geth\\lightchaindata cache=16.00MiB handles=16  
INFO [09-16|23:52:23.721] Opened ancient database database=C:C:\\Users\\jsl\\Code\\201711111\\eth\\geth\\lightchaindata\\ancient/chain readonly=false  
INFO [09-16|23:52:23.733] Writing custom genesis block  
INFO [09-16|23:52:23.739] Persisted trie from memory database nodes=0 size=0.00B time=0s gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B  
INFO [09-16|23:52:23.754] Successfully wrote genesis state database=lightchaindata hash=5704d0..9bc5b0

### 단계 3: 계정 발급

사적망이 생기면 맨 먼저 해야할 일은 뭘까?

네트워크를 시작하려면, 새로운 계정을 만들어야 한다. 계정은 곧 다음 장에서 자세히 설명한다.

명령창에서 계정을 생성해보자. account new 명령어를 사용하고 datadir은 앞서 설정한 디렉토리와 **반드시 동일**해야 한다.

명령어를 입력하면, 비밀번호를 묻게 되는데 이 비밀번호는 꼭 기억해야 한다. 비밀번호를 타이핑할 때, 커서가 움직이지 않으므로 입력이 제대로 되는지 묻는 경우가 많다. 걱정하지 말고 비밀번호를 두 번 입력해서 확인해본다.

비밀 번호까지 입력하면, 통장의 계좌번호 역할을 하는 계좌가 생성이 되는데 상당히 복잡하여 외워서 사용하기 힘들 정도로 암호화되어 있다.

pjt\_dir> geth --datadir .\eth account new  
INFO [09-17|17:17:46.486] Maximum peer count ETH=50 LES=0 total=50  
Your new account is locked with a password. Please give a password. Do not forget this password.  
Passphrase: 여기에 비밀번호를 입력한다.  
Repeat passphrase: 방금 입력한 비밀번호를 재입력한다.  
  
Your new key was generated  
  
Public address of the key: 0x8E6A037be185a70C101cDE94079a0d244Cc30FbA

또는 다음과 같이 geth에 접속한 후, 콘솔 프롬프트에서 명령어를 입력하여 계정을 발급해도 된다.

geth> personal.newAccount("비밀번호")

### 단계 4: geth 사설망 접속

#### geth 사설망 시작

매우 단순하게 다른 설정없이 --datadir에 앞서 정한 디렉토리만 적어서 접속해보자.

pjt\_dir> geth --datadir .\eth

여기서 마치려면 키를 눌러주어서 분명하게 끝내주어야 한다.

콘솔 프롬프트를 원하면 맨 뒤에 console이라고 붙여준다.

pjt\_dir> geth --datadir .\eth console  
> 좌측의 geth 프롬프트가 이렇게 출력된다. 그러나 여기에 로그가 지속적으로 출력되면서 방해가 되기 쉽다.

#### geth의 설정

geth --help를 입력하면, 설정을 위하여 사용할 수 있는 스위치가 상당히 많이 있다는 것을 알게 된다. 버전에 따라 명령어가 변경되는 경우가 있으니 여기서 보인 것처럼 해서 제대로 실행이 안되면 도움말을 확인해서 옵션을 지정한다.

다음 설정을 배우고, 이를 반영하여 필요한 스위치를 사용하여 geth를 시작하도록 하자.

우선 아래 설정에서 접근 방식을 유의해서 배워보자. geth에 접근하는 방식은 rpc, ipc 또는 ws (Web Socket)가 가능하다.

* ipc (Inter-Process Communication)는 **단일 컴퓨터의 운영체제에서**의 프로세스 사이의 호출. 프로세스A가 IPC 파이프 ($HOME/.ethereum/geth.ipc)를 생성, 프로세스B는 이 파이프를 통해 통신한다.
* rpc (Remote Procedure Call)는 컴퓨터와 컴퓨터 사이에서 네트워크를 이용해서 다른 컴퓨터에 있는 프로세스를 간에 네트워크를 통한 프로세스 호출로서, 원격에서 RPC endpoint (localhost:8545) 통신. 일반적으로 TCP, HTTP 프로토콜을 사용한다.
* 마지막으로, ws는 web socket의 통신이며, 최근 geth 버전에서는 이벤트를 사용하기 위해 ws로 원격접속을 설정해야 한다.

아래 사용가능한 스위치를 살펴보면, rpc (버전 1.10부터 http로 교체), ws로 시작하는 옵션을 여럿 발견할 수 있다.

원격접속을 허용하려면 rpc 옵션을 명시해 주어야 한다. 그 방식을 rpc 또는 websocket으로 설정할 수 있다. 원격에서 온라인으로 연결할 때 사용하는 방법인데, 웹페이지에서 geth를 사용하는 것이 좋은 예이다.

여기서는 rpc방식의 접속을 허용하며, 사용할 수 있는rpcapi 또는 wsapi를 db, eth, net, web3, personal와 같이 명시할 수 있다. 즉 원격에서 geth의 db, eth, net, web3, personal 모듈을 사용할 수 있도록 허용하게 된다. 예를 들어, personal은 보안이 필요한 모듈일 수 있고, 사용하지 못하도록 제거하면 된다.

또한 접속을 허용하는 도메인을 rpccorsdomain 또는 wsorigins을 적을 수 있는데, 여기서는 와일드카드 \*로 적어서 제한을 두지 않고 있다. 실제는 허용되는 도메인을 한정하는 것이 보안측면에서 필요하다.

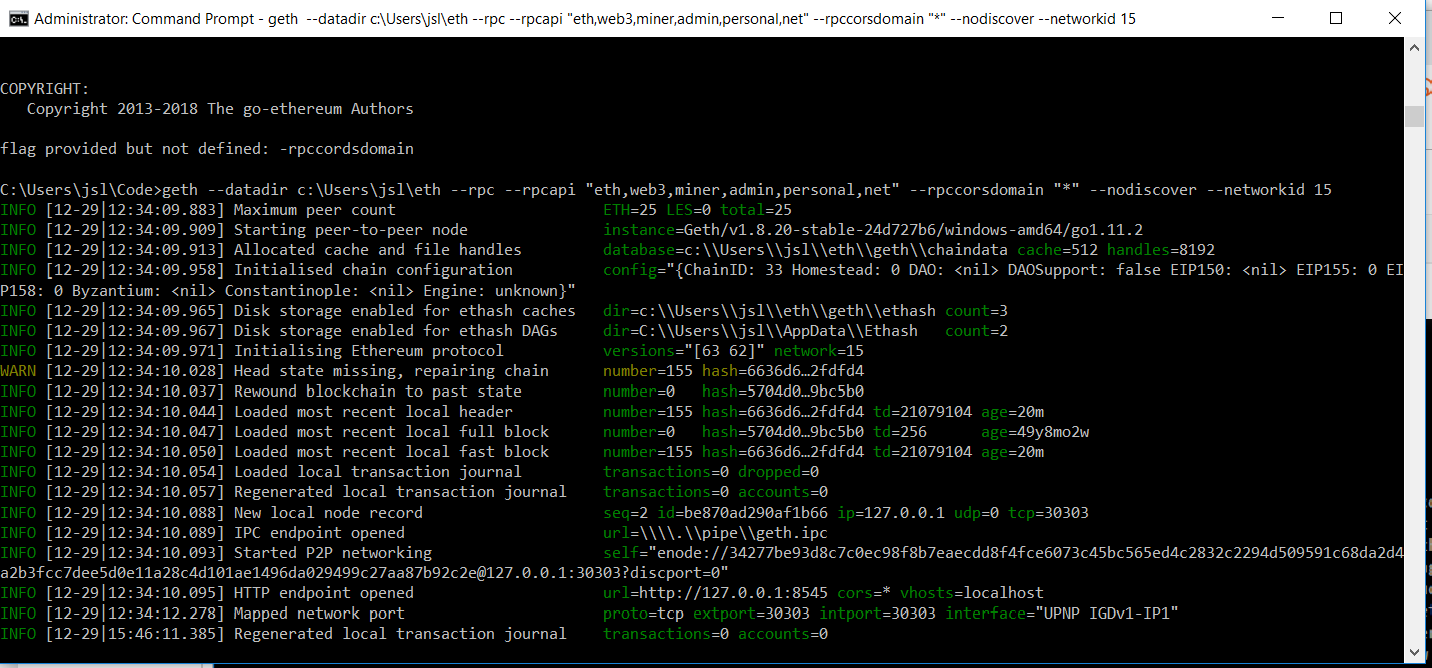
rpcaddr는 자신의 컴퓨터에서 구동할 경우, **ip 대신 --rpcaddr="localhost"**라고 적는다.

더 알아보기: **localhost**

localhost는 로컬 컴퓨터의 IP를 대신해 붙여진 명칭으로서 표준적으로 쓰이고 있다. localhost는 외부와의 통신이 없이 로컬컴퓨터 자신과 통신할 때, 루프백 loopback 주소로 사용한다. IP 주소 127.0.0.0 ~ 127.255.255.255이 할당되어 있다. localhost는 127.0.0.1으로 사용해도 된다.

|  |  |
| --- | --- |
| 선택 | 설명 |
| --networkid | 네트워크 번호, 임의의 양수 0=Olympic, 1=Frontier, 2=Morden. 5이상부터는 genesis block의 ChainID와 동일하게 한다. |
| --identity "jslNode" | 자신의 사설망 명칭. 자신이 인식할 수 있는 명칭을 정해서 적어도 된다. |
| --syncmode | 동기화 방식을 지정 fast, full, or light (1.10버전에서는 snap 가능) |
| --nodiscover | 다른 노드가 자동으로 찾지 못하게 함. |
| --maxpeers | 다른 노드의 최대 접속 수. 0으로 하면 다른 노드를 허용하지 않음. |
| --unlock | 계정의 잔고를 사용할 수 있게 잠금 해제 |
| --rpc | HTTP를 통해 rpc 서버로 사용 (버전 1.10에서는 http) |
| --rpcapi | rpc에서 사용할 api (예: db,eth,net,web3) (버전 1.10에서는 http.api) |
| --rpcaddr | rpc 주소, 0.0.0.0 is all interfaces (버전 1.10에서는 http.addr) |
| --rpcport | 원격접속을 위한 JSON-RPC 포트, 기본값은 **8545** (버전 1.10에서는 http.port) |
| --rpccorsdomain | rpc 요청이 허용되는 도메인 (URL) "\*"는 어떤 도메인이나 허용 (버전 1.10에서는 http.corsdomain) |
| --ws | 웹소켓을 사용. 이벤트의 비동기적 알림이 필요한 경우 사용. |
| --wsapi | ws에서 사용할 api. 생략하면 default만 가능, 주요 기능은 rpc를 사용할 경우 생략해도 된다. |
| --wsaddr | ws 주소. 적어주지 않으면 localhost |
| --wsport | ws 포트, 기본값은 **8546** |
| --wsorigins | 웹소켓 요청이 허용되는 도메인 (URL) "\*"는 어떤 도메인이나 허용 |
| --datadir | 데이터를 저장하는 디렉토리 |
| --port | 네트워크 포트, 기본 값은 **30303** 앞 rpc의 port와 겹치지 않나 생각할 수 있지만 아니다. rpc는 tcp네트워크에서 돌아가는 하나의 서비스 (OSI 7계층), 동일 기계 또는 다른 기계의 프로시저를 호출하는 서비스이다. 반면에 여기서의 port는 tcp에 할당된 포트 (udp도 동일한 포트를 사용, enode에 사용된 포트는 이것을 사용) (OSI 4계층에 해당) |
| --verbosity | Geth 작동 상황의 출력 정도, 0=silent, 1=error, 2=warn, 3=info, 4=core, 5=debug, 6=debug detail |
| --mine | 마이닝 가능 설정 |

#### 실행파일로 geth 시작

 사설망에 접속하는 명령을 스크립트(또는 배치 파일)로 만들면 편리하다.

이런 저런 스위치를 추가하다 보면, geth 실행 명령어는 매번 타이핑을 하기 번거로울 수 있다. 명령어를 저장해서 필요할 경우 편리하게 구동하게 할 수 있다. 사설망을 초기화 설정하는 작업은 실행파일로 저장해 놓으면 편리하다. 사설망은 딱 한 번만 실행하는 것이 아니라, 배워나가면서 망을 올렸다 내렸다 반복하기 마련이기 때문이다.

##### 리눅스/맥 OS 버전(파일 이름: \_gethNow.sh)

아래는 리눅스 쉘문법에 따라 작성한 사설망을 구동하는 명령어이다. 편리한 편집기를 사용해서 작성을 해도 좋다. 단 윈도우 메모장은 보이지 않는 BOM문자가 말썽을 부릴 수 있으므로 피하는 편이 좋다.

유닉스 계열의 운영체제(리눅스 또는 맥os)에서 로그를 조용하게 파일로 보내려고 하는 경우, 리디렉션(redirection) 명령어를 추가할 수 있다. 아래의 맨 뒤 명령어 2>> 파일명는 명령어에서 발생하는 오류 (0번은 STDIN, 1번은 STDOUT, 2번은 오류)를 파일로 **리디렉션**하는 명령어이다. 부등호 > 는 기존 파일의 내용을 지우고 새로 작성하라는 것이며, >>는 기존 내용 뒤에 새로운 내용을 추가하는 명령이다.

공중망에서 사용했던 --synchmode 스위치는 사설망에서는 동기화할 필요가 없으니 굳이 적지 않아도 된다. 혹시 동기화 문제가 발생하면서 로그에 "Switch sync mode from fast sync to full sync" 경고가 뜨는 경우가 있을 수 있다. 그럴 경우 --syncmode snap으로 설정을 추가할 수 있다.

버전 1.10 이상을 사용한다면, rpc 대신 http로 교체되었으니, 아래에 보인 것처럼 rpc 대신 http를 적어준다.

--http --http.addr "xxx.xxx.xxx.xxx" --http.port "8445" --http.corsdomain "\*" \  
--http.api "admin,db,eth,debug,miner,net,shh,txpool,personal,web3" \

[파일명: \_gethNow.sh]  
\_dir=$HOME/eth/  
\_log=$HOME/eth/my.log  
geth --identity "jslNode" \  
--syncmode fast \  
--allow-insecure-unlock \  
--rpc --rpcaddr "xxx.xxx.xxx.xxx" --rpcport "8445" --rpccorsdomain "\*" \  
--rpcapi "admin,db,eth,debug,miner,net,shh,txpool,personal,web3" \  
--ws --ws.addr="xxx.xxx.xxx.xxx" --ws.port "8446" --ws.origins="" \  
--datadir $\_dir \  
--port "38445" \  
--networkid 33 \  
console 2>>$\_log 명령창에 콘솔을 생성하고, 오류를 로그로 재지정하는 명령어, 이 줄은 생략해도 된다.

리눅스는 쉘로 만들어서 다음과 같이 실행한다. 파일의 확장자가 .sh로 끝나고 있고, 이런 쉘 파일을 실행하기 위해서는 sh라고 앞에 적어준다.

pjt\_dir> sh \_gethNow.sh

##### 윈도우

윈도우에서 하려면 약간만 수정하면 된다. 거의 유사하다. 디렉토리 명칭은 알맞게 수정하자.

* 줄2 \_log는 로그파일을 생성하지 않으므로 삭제해도 된다.
* 줄3부터 끝에 쓰인 ^는 캐럿 Caret이라고 하는데, 긴 명령어를 읽기 좋게 여러 줄로 나눌 때 사용하는 기호이다.

[파일명: \_gethNow.bat]  
set \_dir=C:\Users\jsl\Code\201711111\eth  
set \_log=C:\Users\jsl\Code\201711111\eth\my.log  
geth --identity "jslNode" ^  
--syncmode snap ^  
--allow-insecure-unlock ^  
--http --http.addr "localhost" --http.port "8445" --http.corsdomain "\*" ^  
--http.api "admin,db,eth,debug,miner,net,shh,txpool,personal,web3" ^  
--ws --ws.addr="localhost" --ws.port "8446" --ws.origins="" ^  
--datadir $\_dir ^  
--port "38445" ^  
--networkid 33

윈도우에서는 배치프로그램으로 실행하기 때문에 .bat로 저장해서 실행하도록 한다. 리눅스와 달리 명령어 프롬프트에서 그냥 파일명 \_gethNow.bat을 적고 하면 된다.

pjt\_dir> \_gethNow.bat

몇 가지 주의할 것이 있다.

* --unlock 0는 자신이 가지고 있는 첫번째 계정을 잠금 해제하는 설정이다. 송금이나 지급이나 어떤 거래에는 수수료가 따라 붙는다. 계정을 해제해 놓지 않으면, 수수료를 포함하여 출금이 허락되지 않는다. 매 거래마다 계정을 풀어놓을 수 있지만, 아예 풀어 놓으면 위험하기는 하겠지만 (사설망에서는 가치를 가지고 있지 않으니 굳이 위험하지도 않겠다), 간편하므로 이렇게 해제해 놓으면 편리하다. 이런 계정해제 스위치는 rpc 또는 ws를 활성화하면 사용할 수 없다. unlock을 시도하면 "Fatal: Account unlock with HTTP access is forbidden!" 오류 메시지가 발생한다. 대신에 --allow-insecure-unlock를 사용하자.
* 또한 앞서 **비밀번호를 완전히 해제 (unlock)해 놓은 경우, --rpccorsdomain="\*" (또는 --http.corsdomain)이라고 하지 않도록 한다**. 비밀번호가 해제되어 있으면 어디서든지 접근가능하기 때문에 안전을 보장할 수 없다.
* 위에서 적은 networkid는 우리가 앞서 생성한 \_genesis.json에서 설정해 놓은 chainID와 동일하게 한다.
* rpcport와 port는 임의로 번호를 설정할 수 있지만, 적은 숫자는 시스템에서 사용하고 있으므로 아래 명령문에서 쓰인 정도로 적당히 큰 번호를 사용하도록 하자. --rpcapi에 허용하는 api를 적어준다.
* 웹소켓을 설정한다. web3js를 사용하면서 web3.eth.subscribe하는 경우 필요하다. 이 함수는 이벤트가 발생하면 로그에 기록하고, 이러한 이벤트의 발생을 비동기적으로 알려준다. geth는 이러한 push 알림을 해주지 않기 때문에, 웹소켓이 필요하다. --wsapi는 --rpcapi와 동일하게 설정해도 되나, 이벤트 알림을 위해서 사용하는 경우 생략해도 된다. 나중에 이벤트를 발생하는 것은 별도로 배우게 된다.

geth가 실행되면 (\_gethNow.bat 또는 \_gethNow.sh를 실행), 다음과 같이 웹소켓이 추가된 로그를 볼 수 있다.

INFO [05-19|06:45:53.638] IPC endpoint opened url=/home/jsl/eth/geth.ipc  
INFO [05-19|06:45:53.639] HTTP endpoint opened url=http://xxx.xxx.xxx.xxx:8445 cors=\* vhosts=localhost  
INFO [05-19|06:45:53.639] WebSocket endpoint opened url=ws://xxx.xxx.xxx.xxx:8446

#### geth 사설망 접속

geth 사설망이 시작된 후에 사설망에 접속해본다.

주의! geth 사설망은 시작되었지만 console 프롬프트가 없다. 즉 백그라운드로 실행되고 있는 것이다. 따라서 접속창을 attach로 만들어서 콘솔 작업을 가능하게 해야 한다.

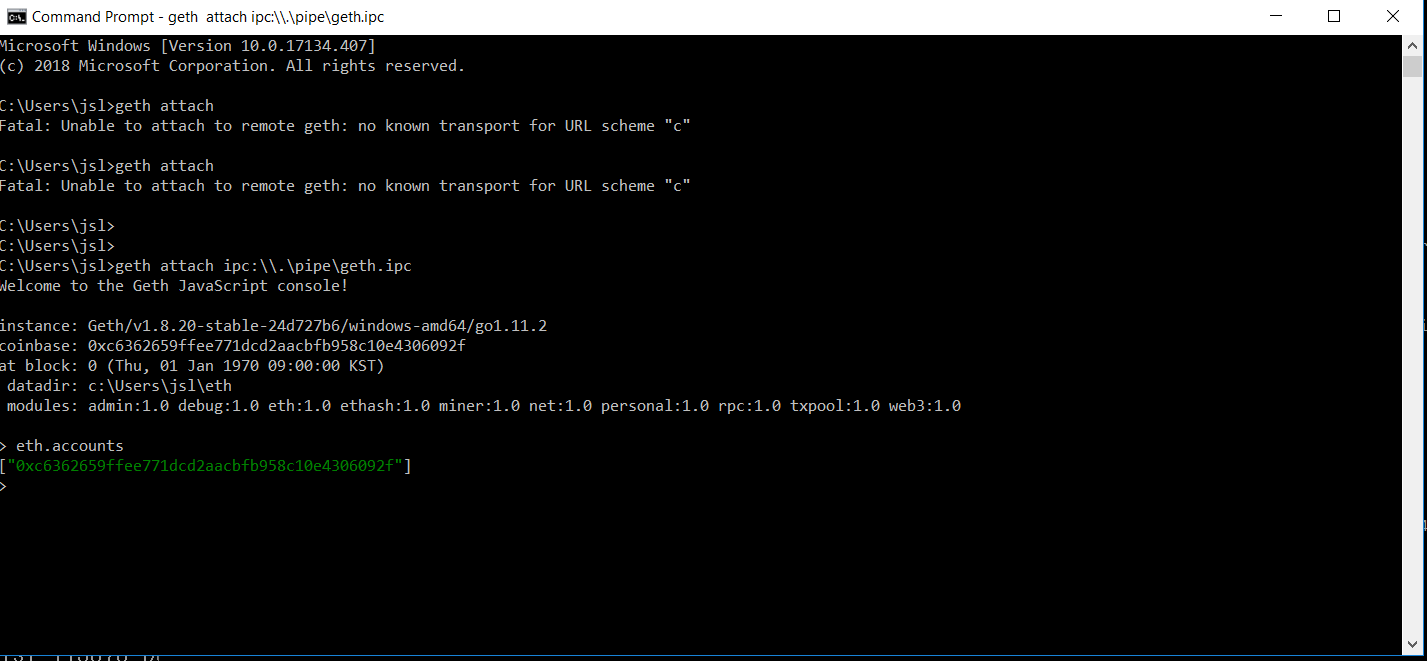
스크립트에서는 geth 명령을 실행시키면서 console을 지정하지 않았다. 앞서 설명한 것처럼 console을 붙여서 실행하면, geth에서 발생하는 로그 내용들이 계속 화면에 출력되기 때문에 입력 작업이 쉽지 않다.

이럴 때는 geth attach 명령어를 사용하면 로그 내용을 보지 않고 geth 작업을 할 수 있다.

ip, rpc, ws에 다음과 같이 연결할 수 있다.

geth attach ipc:/some/custom/path  
geth attach http://localhost:8445 <--- 책에서는 주로 http 연결을 한다  
geth attach ws://localhost:8446

여기서는 잘 쓰지 않지만, 다음 화면은 ipc로 연결하고 있다.



alt text

geth console에서는 다음에 보인 것처럼 실행해도 된다.

geth> admin.startRPC("ip.address",8445,"\*","web3,db,net,eth")

접속이 올바르게 되었는지 geth 콘솔 프롬프트에서 확인할 수 있다.

geth> admin.nodeInfo.protocols  
{  
 eth: {  
 config: {  
 chainId: 33,  
 eip150Hash: "0x0000000000000000000000000000000000000000000000000000000000000000",  
 eip155Block: 0,  
 eip158Block: 0,  
 homesteadBlock: 0  
 },  
 difficulty: 93425390,  
 genesis: "0x5704d029fe80f4fb605c0cb5e31d591511f10a46a0cb8166f97d8d559f9bc5b0",  
 head: "0x1bf49142fd5666dc16ed51961eb3b82351a6b403f511a6d3a2d9db0d84678d2f",  
 network: 33  
 }  
}

개설된 데이터 디렉토리의 구성을 살펴보자. geth를 실행시켰던 datadir인 eth에 다음에 보인 하위 디렉토리들 geth, keystore가 생성되어 있다.

리눅스의 경우, eth/geth 디렉토리 아래는 다음 파일이나 디렉토리가 생성되어 있다. 윈도우도 비슷할 것이다.

pjt\_dir> ls -l ./eth/geth 또는 윈도우 dir  
total 28  
drwxr-xr-x 2 jsl jsl 4096 3월 11 22:30 chaindata 블록체인 데이타  
drwxr-xr-x 2 jsl jsl 4096 1월 3 16:57 lightchaindata  
-rw-r--r-- 1 jsl jsl 0 1월 3 16:59 LOCK  
-rw------- 1 jsl jsl 64 1월 3 16:59 nodekey 공적키. 파일크기가 64바이트이다.  
drwxr-xr-x 2 jsl jsl 4096 3월 11 22:30 nodes 피어노드의 접속 데이터  
-rwxr-xr-x 1 jsl jsl 8538 3월 11 22:30 transactions.rlp 거래 목록이 저장.

eth/keystore 디렉토리에는 사적키가 저장되어 있다.

pjt\_dir> ls -l ./eth/keystore/  
total 8  
-rw------- 1 jsl jsl 491 1월 4 06:57 UTC--2019-01-03T21-57-03.966937024Z--21c704354d07f804bab01894e8b4eb4e0eba7451  
-rw------- 1 jsl jsl 491 1월 4 10:40 UTC--2019-01-04T01-40-01.674920065Z--778ea91cb0d0879c22ca20c5aea6fbf8cbeed480

### 단계 5: 충전

새로운 계정은 당연히 잔고가 '0'으로 되어 있다. 잔고가 없으면 거래를 할 수 없다.

계정이 발급하고 나서는 이더(Ether)를 획득해야 한다. 충전을 통해서 Ether를 획득한다.

사적망 Ether는 화폐가치가 있을까? 공중망에서만 화폐가치가 있고, 테스트 및 사적망에서 생성되는 이더는 화폐가치가 없다.

계정은 방금 만들었으니, 당연히 잔고가 없을 것이다. 잔고가 없으면 거래를 발생할 수 없게 된다. 무엇보다 먼저 계정에 충전을 해야하며, 마이닝을 해서 충전하도록 한다.

사설망에서는 굳이 충전하기 위해 무료로 테스트용 이더리움을 받을 수 있는 faucet 서비스를 이용할 필요가 없다.

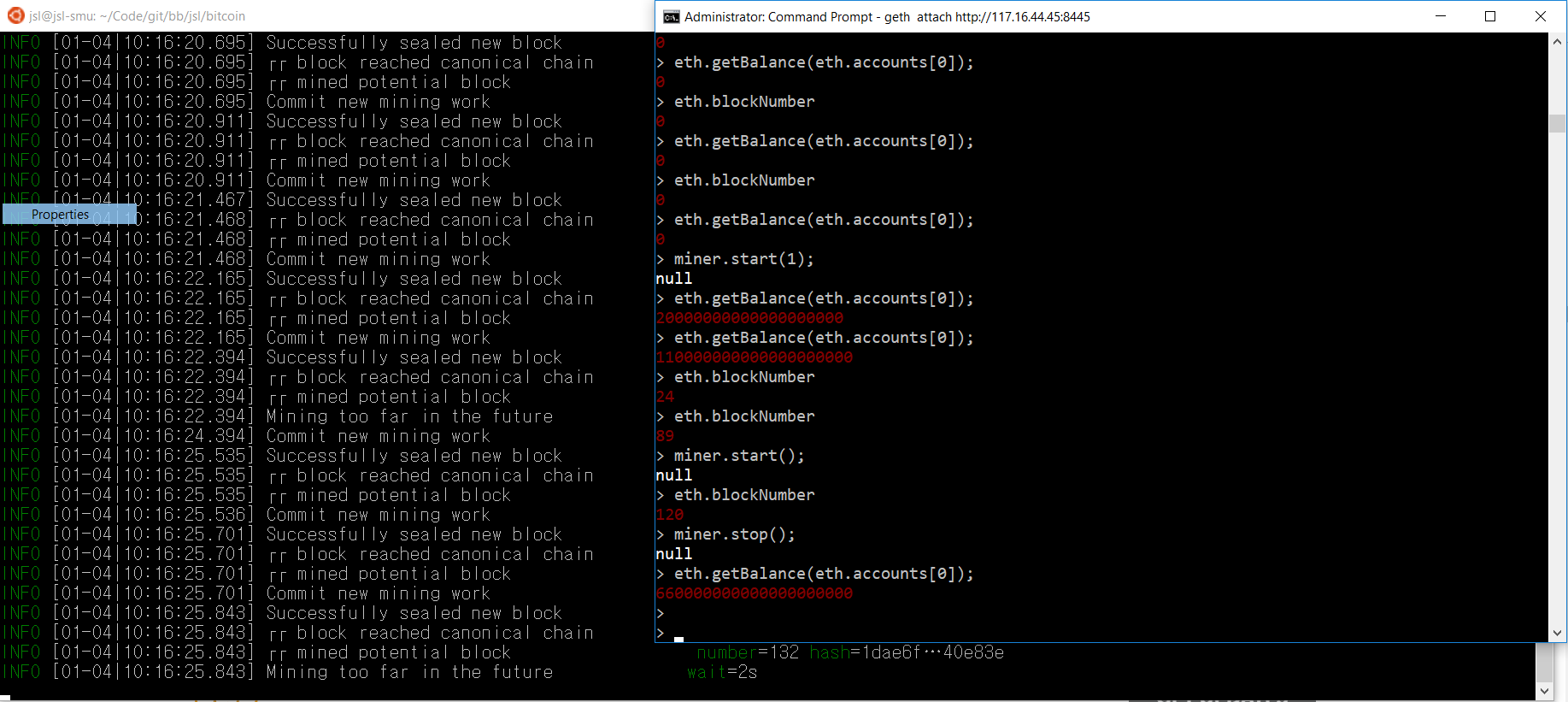
geth console에서 다음 명령을 실행시켜 마이닝을 시작한다.

geth> miner.start()

적당한 시간이 지나면 마이닝을 종료 miner.stop()할 수 있다. 5분 정도만 충전을 해도 실제화폐 가치로는 상당한 금액이 계좌에 입금되어 있을 것이다.

geth console에서 eth.getBalance(인자는 coinbase를 적음)를 입력하면 생성된 Ether를 출력한다.

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

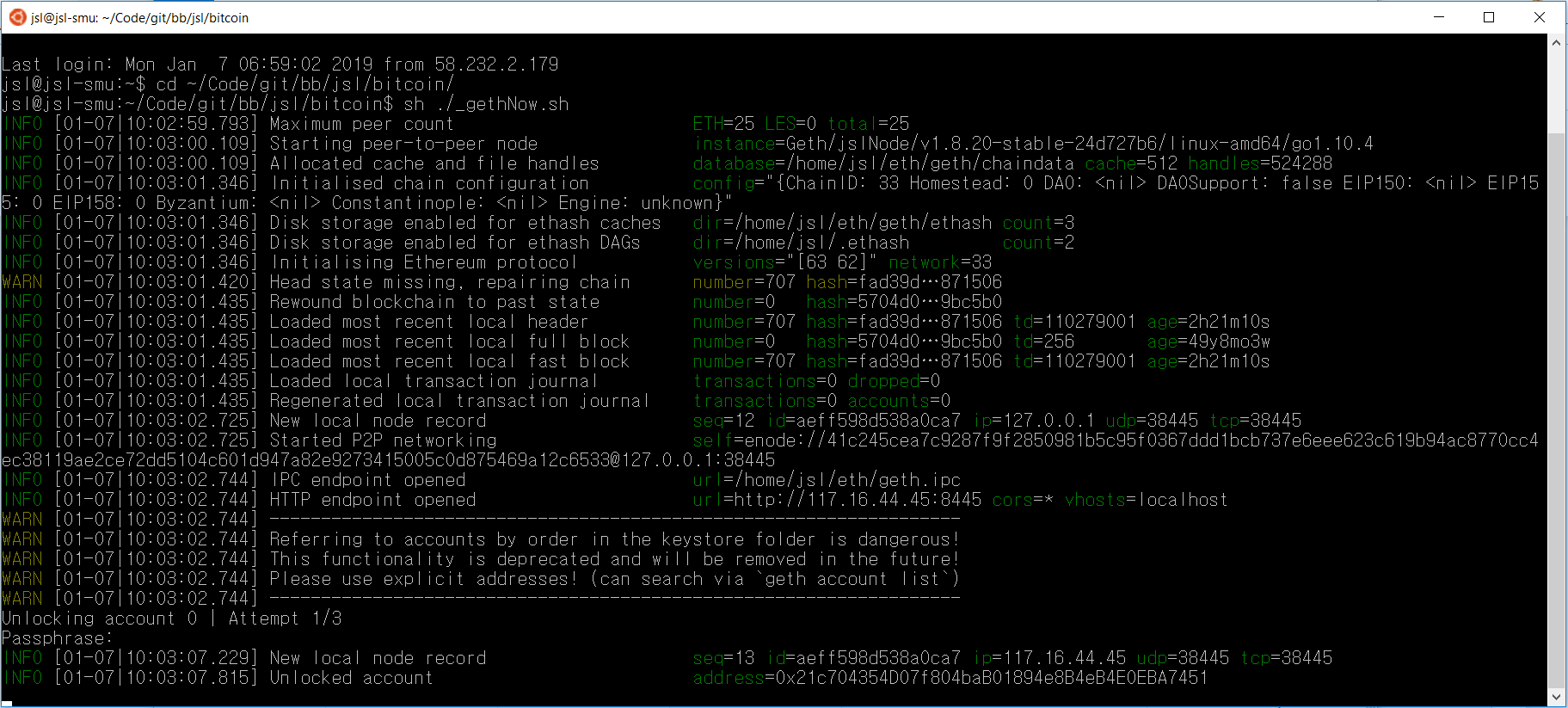


alt text

### 단계 6: geth 종료하기

geth를 올바르게 종료하지 않으면, 지난 세션의 동기화를 잃어버리게 된다.

즉 윈도우를 강제로 닫거나, 원격에서 열어놓은 단말이 네트워크 연결이 끊어져 올바르게 geth 세션이 종료하지 않는 경우 그런 동기화를 모두 잃어버리게 되는 상황이 발생한다. 올바르게 geth를 종료하려면

* 백그라운드로 실행되고 있는 경우 Ctrl+C 그러면 맨 뒷 줄에 Database closed라고 출력하면서 종료한다.
* 단말이 열려 있는 경우 geth 프롬프트 >에서 마치려면 exit을 입력한다.
* pjt\_dir> geth --datadir .\eth console  
  ...생략...  
  INFO [09-24|17:10:03.651] Allocated cache and file handles database=C:\\Users\\jsl\\Code\\201711111\\eth\\geth\\chaindata cache=512 handles=8192  
  ...생략...  
  > exit  
  INFO [09-24|17:10:06.488] IPC endpoint closed url=\\\\.\\pipe\\geth.ipc  
  ...생략...  
  INFO [09-24|17:10:06.577] Blockchain manager stopped  
  INFO [09-24|17:10:06.582] Stopping Ethereum protocol  
  INFO [09-24|17:10:06.589] Ethereum protocol stopped  
  INFO [09-24|17:10:06.594] Transaction pool stopped  
  INFO [09-24|17:10:06.602] Database closed database=C:\\Users\\jsl\\Code\\201711111\\eth\\geth\\chaindata
* 
* alt text

종료한 뒤에 다시 사설망에 접속해서 채굴된 이더 등이 지속적으로 남아 있는지 확인해본다.

## 문제 3-1: IP기반 사설망으로 원격접속

앞에서 사적망을 구성하였다. 이를 컴퓨터A라고 하고, 여기에 고정 IP 주소를 적도록 한다.

통신사에서 제공하는 인터넷을 사용하게 되면, 고정 IP를 사용하지 않는다. 반면에 학교나 기업과 같은 기관에서는 고정IP를 부여받을 수 있다.

고정 IP를 사용하면, 그 IP를 기반으로 geth를 띄워 놓을 수 있다. 이렇게 하는 것은 클라이언트-서버 방식에서, IP기반의 서버를 구동해 놓는 것과 유사하다. 이런 컴퓨터A에 대해 다른 컴퓨터B에서 원격접속을 할 수 있다.

### 단계1: Geth 서버를 실행

자신이 IP주소를 부여받았는지 확인하자. 앞서 만들어 놓은 \_gethNow.bat (또는 리눅스의 \_gethNow.sh)에 그 IP가 적혀 있어야 한다.

그 IP주소의 컴퓨터A(서버 역할)에서 geth 서버를 실행한다.

### 단계2: 원격접속

다른 컴퓨터B로 이동한다. 이를 클라이언트라고 하자. 클라이언트 컴퓨터B에서 geth attach를 실행한다. 이 때 IP주소를 적어주어야 한다.

클라이언트에서의 원격접속은 로컬에서 해도, 원격에서 해도 된다.

여기서는 컴퓨터A에 서버역할의 geth를 백그라운드로 실행하고, 컴퓨터B(윈도우)에서 원격접속하고 있다.

컴퓨터B> geth attach http://xxx.xxx.xxx.xxx:8445  
Welcome to the Geth JavaScript console!  
  
instance: Geth/jslNode/v1.9.9-stable-01744997/linux-amd64/go1.13.4  
coinbase: 0x21c704354d07f804bab01894e8b4eb4e0eba7451  
at block: 49277 (Thu, 20 May 2021 13:29:08 KST)  
 datadir: /home/jsl/eth  
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0  
  
> eth.blockNumber 원격에서 blockNumber가 서로 일치하는지 확인해본다.  
49277

## 문제 3-2: 멀티노드 사설망 구성하기

### 특정 컴퓨터를 Peer로 추가하여 멀티노드를 구성해 보자

사적망은 개인적인 네트워크인 것처럼 보이지만, **동일한 네트워크 ID를 가진 노드**라고 하면, 누구라도 Peer로 참여될 수 있다. 물론 초기화를 위한 genesis.json 내용이 동일해야 하기 때문에 충돌할 가능성은 매우 적다. 그렇게 되면, 사적망에 자신의 컴퓨터만 연결된 것이 아니라, 여러 컴퓨터가 피어 노드로 참여하게 된다는 점에서 점에서 엄밀히 사적망은 아닐 수 있다.

사적망이라 하더라도 멀티노드는 서로 동기화 한다. 블록체인을 서로 동기화되기 때문에 blockNumber가 동일하게 된다.

특정 컴퓨터를 지정하여 노드로 참여시키고 클러스터를 구성해 보자. 자신을 노드A라고, 참여하는 측을 노드B라고 하자.

### 단계 1: 멀티 노드 설정 및 실행

멀티노드 환경에서는 모든 노드가 동일한 블록체인 상태를 가지고 있어야 하므로, Peer로 추가하려면 노드A와 노드B의 **genesis block**을 동일하게 설정해 놓아야 한다. **networkID** 역시 동일하게 설정한다. admin.nodeInfo.protocols.eth의 값을 서로 비교해 보자. 또한 web3.eth.getBlock(0)의 hash가 서로 동일한지 비교해 보자.

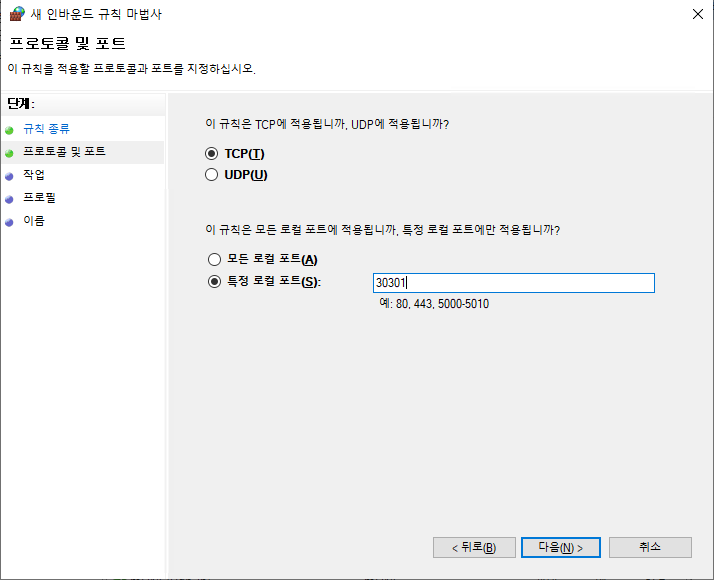
Peer는 **정적 IP**를 가지고 있고, 외부에서 자신의 컴퓨터를 찾을 수 있어야 한다. 자신의 IP에 대해 특정 포트가 열려 있는지 확인할 수 있는 사이트 https://canyouseeme.org/ 를 방문하여, 예를 들어 포트번호 30301를 넣어 확인한다.

**genesis block이 동일하여도 Peer로 추가할 수 없다면**, 네트워크 문제가 있어 연결이 안되는 것이다. **firewall**이 막는 경우 특히방화벽이 있는 경우 Peer를 찾을 수 없다. --port가 개방되어 있어야 한다. 우리의 경우 30301로 설정되어 있다.

리눅스에서는

sudo ufw allow 30301

윈도우에서는 방화벽을 열어서 포트 30301을 열어준다.



윈도우 방화벽 포트 지정

또는 **운영체제 시간**이 맞지 않는 경우 동기화 시킨다.

sudo ntpdate -s time.nist.gov

**Peer** 찾기

P2P 시스템에서는 Peer를 찾기 위한 알고리즘이 필요하다. **Port/IP scanning**은 가능한 모든 IP, Port에 접근해서 응답하는지 찾아보는 방식으로, 낭비가 매우 많다. 보통 후보 Peer가 적힌 목록을 가지고 연결할 수 있다. 또는 처음 시작할 때 연결할 bootstrap nodes를 설정할 수 있다. (컴퓨터에서의 부팅) 연결되는 노드라 **bootnode**라고도 한다. 네트워크를 시작하는 시점에 'geth --bootnodes'에 나열해서 특정 peer를 static node로 연결할 수 있다. 이더리움에서는 Kademlia 알고리듬으로 Peer를 찾는다.

### 단계 2: enode값을 구한다.

enode는 이더리움 네트워크 상에서 컴퓨터를 찾기 위해 필요한 URL과 유사한 역할을 한다. enode에는 네트워크 연결에 필요한 IP 주소, 포트번호를 포함하고 있다. 추가하는 노드B의 geth console에서 명령을 실행시켜 enode를 찾는다.

geth> admin.nodeInfo.enode  
"enode://e0a396df0b5f68d20...5e8f0c@[::]:30446"

### 단계 3: 노드 추가

노드A에서는 위에서 찾은 노드B의 enode 값을 추가해야 Peer로 인식하게 된다.

잠깐, enode 값에 IP가 빠져있다. 여기 [::] 부분에 IP주소를 적어주어야 한다.

추가하는 방법은 (1) 필요시 또는 (2) 고정적으로 할 수 있다.

우선 필요시 추가하기 위해서는 addPeer() 함수를 이용한다. 노드A의 geth 프롬프트에 아래와 같이 함수 인자에 노드B의 enode를 적어주면 된다.

geth> admin.addPeer("enode://719b29cc599ee5964...11155676a3cb@xxx.xxx.xxx.xxx:30446?discport=0")  
true

고정적으로 하려면, 설정에 넣어야 한다. 아래에서 보듯이, 노드 2개의 enode를 적어주고, 시작과 끝 대괄호를 넣어서 json 형식으로 쓴다.

[파일: static-nodes.json]  
[ "enode://e0a396df0b5f68d209c7fa3...1fd1ed4a4429342175c7aab9e415e8f0c@xxx.xxx.xxx.xxx:30446",  
"enode://00c177125f949f51290b5e4...558326564300e7a9d3809a1c15ef97566@xxx.xxx.xxx.yyy:30445"  
]

설정 파일 static-nodes.json는 --datadir에 적었던 디렉토리 밑에 다음과 같이 저장해 놓는다. 그 디렉토리 밑에는 keystore 디렉토리가 있고, 그와 같은 수준에 저장하면 된다.

pjt\_dir\eth  
 |  
 -- keystore\  
 |  
 | static-nodes.json

또는 geth 실행파일에 enode를 적어주어도 된다. 앞서 작성한 \_gethNow.bat 또는 \_gethNow.sh를 기억할 것이다. 그 파일의 --bootnodes에 enode를 다음과 같이 추가해주면 된다.

geth --identity "nameNode" \  
 ...생략...  
 --bootnodes "enode://e0a396df0b...생략...9e415e8f0c@xxx.xxx.xxx.xxx:30446" \  
 --maxpeers 1 \ 기본 값은 0이다.  
 ...생략...

멀티노드를 성공적으로 구성하면, peer접속 정보를 볼 수 있다.

geth> admin.peers  
[{  
 caps: ["eth/62", "eth/63"],  
 id: "e0a396df0b...생략...9e415e8f0c", 추가한 enode와 동일한 값을 가지고 있다.  
 name: "Geth/jslNode/v1.5.0-unstable/linux/go1.6.2",  
 network: {  
 localAddress: "xxx.xxx.xxx.xxx:45034",  
 remoteAddress: "xxx.xxx.xxx.xxx:30446" 추가한 IP, 포트와 동일한 값을 가지고 있다.  
 },  
 protocols: {  
 eth: {  
 difficulty: 218833674473,  
 head: "0x08f0271d893ee08ea387773c10e672a26ad9b8a2dd19dcbd503b859b4f0bab0e",  
 version: 63  
 }  
 }  
}]

## 연습문제

1. 이더리움 네트워크를 개방정도에 따라 구분한다면, 그 명칭을 적으시오.
2. 네트워크 ID, 1번과 3번은 어떤 망에 할당되어 있는지 적으시오.
3. 공중망이나 사적망이나 식별하기 위해 ID가 부여되고 있다. OX로 답하시오.
4. 사적망은 ID가 동일하면 충돌하기 때문에 주의한다. OX로 답하시오.
5. 비트코인이나 이더리움의 Ether는 구입하거나, 마이닝해서 획득할 수도 있다. OX로 답하시오.
6. 공중망에서 사용하는 암호화폐 Ether는 실제화폐 가치가 없다.OX로 답하시오.
7. 공중망에 참여하려면 암호화폐 Ether가 있어야 한다. OX로 답하시오.
8. 공중망은 분산자원을 사용하기 때문에, 자신의 컴퓨터에서 실행해 놓아도 CPU나 메모리 자원은 사용되지 않아도 된다. OX로 답하시오.
9. 마이닝을 하게 되면 혹시 Ether를 획득할 수 있다. OX로 답하시오.
10. 블록체인을 동기화하기 위해서는 상당한 시간이 필요하지만, 완전 동기화하지 않고서는 참여할 수 없다. OX로 답하시오.
11. 원격 접속하면서 허용되는 기능을 명시할 수 있는데, eth, web3, admin 모듈은 제공하는 편이 좋다. OX로 답하시오.
12. rpc 원격접속을 설정하면서 rpccorsdomain를 풀어 놓아도 보안이 약해지지 않는다. OX로 답하시오.
13. 공중망을 실행하고 있으면, 굳이 테스트망은 필요하지 않다. OX로 답하시오.
14. 테스트망의 Ether는 실제 화폐 가치가 있다. OX로 답하시오.
15. 테스트망에서 계정에 잔고를 충전하기 위해서는 구매를 해야 한다. OX로 답하시오.
16. 공중망에 참여하기 위해서는 Ether가 필요하지만, 테스트망에 필요하지 않다. OX로 답하시오.
17. 보통은 애플리케이션을 배포하려면 서버가 있어야 한다. 공중망을 사용한다면 그런 서버가 필요하다. OX로 답하시오.
18. 사적망을 설정없이 쉽게 개설하는 명령어를 적으시오.
19. 사적망 개설할 때 블록체인을 초기화하기 위해 필요한 JSON 파일과 필드를 적으시오.
20. 사적망을 개설할 때 디렉토리 설정을 하지 않아도 문제가 되지 않는다. OX로 답하시오.
21. 사적망을 생성하고 나서, 계정은 기본으로 생성되어 있다. OX로 답하시오.
22. 사적망 Ether는 화폐가치가 있다. OX로 답하시오.
23. 메인네트워크 접속하고 'admin.nodeInfo' 출력하시오.
24. 테스트네트워크 접속하고 'admin.nodeInfo' 출력하하시오.
25. 블록체인 사설망을 개설하시오. Chain ID 305, 데이터 디렉토리는 C:\Users\G305\eth를 사용하도록 설정한다. 사설망에 http 접속하여 geth 클라이언트를 개설하고:

* ChainID가 설정한 번호와 동일한지 확인하고,
* 사설망 개설하고 나서 출력되는 로그를 확인하고,
* 계정을 생성하고, 충전한 후 잔고를 출력한다.