# 블록 체인 : 모든 거래자의 전체 거래장부 공유 및 대조를 통해 거래를 안전하게 만드는 보안 기술

- 기존 거래 방식 : 거래내역을 작성하고 그 내용을 확인 후 거래가 일어남 (은행에서)
➔ 최소한만 저장 , 최소한의 인원만 접근하여 확인 (비밀은 최소한만 알고 보안하는게 안전하다고 생각했기 때문에 은행만 알고 있음)

**- 블록체인 거래방식**: 블록 이 존재( 새로운 거래기록 저장),블록체인으로 연결된 pc 는 10 분 간격으로 거래내역 내용 비교 하고 과반수 이상 동의가 일어나면 블록화! ==> 인증받은 거래내역만 남게 된다 / 인증받지 못한 거래내역 폐기
거래자 거래장부 공유(암호화된)
➔ 따라서 거래내역 위조 어려움 ( 블록체인의 과반수 이상 다 해킹해서 거래장부를 속여야 하는데 불가능)

**- 핵심 기술** : 해시 **Hash (문장 길이에 관계없이 일정한 길이의 값으로 변경)**
➔ 문장 내용이 완전히 같으면 동일한 완전히 같은 해시값 가짐! (하지만 문장이 조금이라도 다르면 완전히 다른 해시값 가짐)
➔ 해시값 조합을 통해 원문을 유추 할수 없음
➔ 적은 데이터양으로 원본내용 모두 완전히 같음을 비교 가능! (원본 내용이 엄청 많더라 하더라도 그 내용을 해시 값으로 변경해서 비교하면 적은 데이터양으로 비교 가능!)

암호화폐 : 블록체인에서 관리하고 있는 화폐(해쉬에 의해서)

**장점: 기록물, 그 권한의 분산화(탈중앙화) ==> 의료, 금융, 물류 등등**

1. 관리 효율성
2. 기록 신뢰, 보안성

**Blockchain 선택(3 가지)**

1) Main Net : Public blockchain 으로 누구나 접근 가능. 한개만 존재
2) Test Net: 자체 Ether 를 발행하여, 테스트 용도로만 사용. 복수의 테스트넷 존재
3) Local Net : 자신의 PC 에 온전한 blockchain 을 올려서 개발, 복수 개 만들어서 하는 것도 가능

**트랜젝션(transactions)**

- 블록체인은 전역적으로 공유되는 트랜젝션 데이터 베이스

- 이것은 누구든 네트워크에 참여하는 사람은 데이터베이스의 엔트리를 읽을 수 있다는 뜻

- 트랜젝션이 의미하는 것은 어떠한 변화를 만들고자 한다면 아무런 변화가 없거나 완전히 반영되어야 함

- 트랜젝션이 데이터베이스에 반영되는 동안 다른 트랜젝션은 이를 바꿀 수 없다.

# 이더리움

- 이더리움은 DApp 을 배포할 수 있는 탈중앙화 플랫폼이다.
- 현재 이더리움 엔진은 Go 언어와 C++, 파이썬 등으로 개발 되어있음
- 가장 활발하게 개발이 진행 되고 있는 것은 Go 로, go-ethereum(줄여서 Geth)
- 이더리움 엔진인 geth 은 3 가지 인터페이스를 통해 활용 가능(HTTP JSON RPC, web3.js, Solidity)
- 실행환경 : Virtual Box 라는 가상화 엔진에 geth 라는 EVM(Ethereum Virtual Machine)이 위치하는 구조
- EVM 은 가상환경이기 때문에 js,파이썬 같은 기존 언어 사용가능

## 1. 설치(Geth : 이더리움의 전체 기능을 사용할 수 있는 풀 클라이언트)

$ brew tap Ethereum/Ethereum
$ brew install Ethereum
$ git clone -b release/1.8 https://github.com/ethreum/go-ethereum.git
$ cd go-ethereum
$ make geth

2) 하위 명령어 및 옵션

- 메인넷 네트워크 연결 : 이더리움 네트워크의 노드들은 기본적으로 30303 포트로 통신
(다른포트도 리스닝 가능)

## 2. go 설치( /usr/local/go )

- https://golang.org/dl/ 에서 osx10.8.pkg 설치
$ sudo vi .bash_profile ➔ ( export PATH=$PATH:/usr/local/go/bin (환경변수) )
$ source ~/.bash_profile
$ go 또는 go env

hello 실습

```
$ vi hello.go
```

다음과 같이 작성해주세요.

```
/* hello.go - My first Golang program */
package main
import "fmt"
func main() {
    fmt.Printf("Hello, world\n")
}
```

자 이제 실행해봅시다.

```
$ go run hello.go
Hello, world
```

## 3. GAS 란?

- 이더리움 플랫폼에서는 이더(ether)라는 자체 화폐토큰이 있고 이더를 가지고 가스(GAS)라는 어플리케이션을 구입해 이더리움이 Smart Contract 를 하는데 연산력과 저장공간 제공의 '연료'로서 쓰이게 된다. 그렇게 되면 명령어에 따라 특정 조건에서 **자동적이고 강제적인 계약**이 이행된다.
➔ 이더리움에서 트랜젝션을 실행시키기 위해 필요한 **수수료!**

**4. Smart Contract 란?**

- 특정 계약 조건을 이행하는 것

**5. geth 네트워크 초기화 / 참고 : http://locallab-seoul.tistory.com/2**

1) 로컬 테스트넷에서 Geth 를 기동하기 위해 데이터 디렉터리**, Genesis 파일 생성**
- Genesis 파일은 블록체인의 0 번째 블록의 정보가 저장되어 있는 JSON 형식의 텍스트 파일
- 데이터 디렉터리 : 송수신한 블록 블록데이터와 계정 정보를 저장,
서로 다른 블록체인 네트워크 사이에서 공유 가능

$ mkdir testnet ( dev/BlockChain_Project )
$ $ geth --datadir testnet init testnet/genesis.json
$ tree ( 명령어 사용하여 초기화된 테스트 네트워크 구성 확인 가능 ) / $ brew install tree

**2) geth 콘솔 실행**

**$ geth –networkid 4649 –nodiscover –maxpeers 0 –datadir dev/BlockChain-Project/testnet/ console >>2 dev/BlockChain-Project/testnet/geth.log**

--networkid : 네트워크 식별자
--nodiscover : 생성자의 노드를 다른 노드에서 검색할 수 없게 하는 옵션
--maxpeers : 생성자 노드에 연결할 수 있는 노드 수
--datadir : 네트워크 데이터 디렉터리
>>2 dev/BlockChain-Project/testnet/geth.log (로그파일이 없다면 만들고 여기에 에러내용 기록)

**3) 계정 생성 및 확인 및 송금 실습 ( EOA )**

➔ 이더리움에는 2 가지 종류의 계정이 있다 EOA, Contract
➔ EOA(Externally Owned Account) : 송금 및 계약 실행 가능 계정 생성 ( 일반사용자 계정, 비밀키로 관리)
➔ Contract 계정은 계약을 블록체인에 배포할 때 만들어지는 계정으로 블록체인에 존재

> personal.newAccount(“”). // 비밀번호 입력
➔ "0xf15fbd765e30660d293a09668bae7ec66aa8b964" (생성된 계정 주소)

> eth.accounts // 생성된 계정 확인

> eth.accounts[0] // 인덱스를 통해서 여러 개정이 있을때 주소 확인 가능

**4) 채굴(Mining)**

> eth.getBalance(eth.accounts[0]) // 첫번째 계정의 잔고 확인

➔ 채굴을 하여 Ether 를 얻을 수 있고 채굴에 대한 보상을 받는 계정을 Etherbase 라고 함

➔기본적으로 제일 처음 생성된 계정(eth.accounts[0])이 Etherbasefh 로 지정

> miner.setEtherbase(web3.eth.accounts[0]) ( 채굴 성공 후 보상받을 계정 설정 가능)

> eth.accounts[0]

> eth.coinbase // eth.coinbase 명령으로 Etherbase 계정 주소를 확인

> miner.start(1) // 채굴 시작 / 입력값 1 은 채굴 시 스레드 개수를 1 개로 설정

> miner.stop() // 채굴 종료

> eth.blockNumber // 채굴한 결과 확인(블록의 갯수)!

> eth.getBalance(eth.accounts[0]) // 첫번째 계정의 잔고를 통해 채굴 보상 결과확인! ➔ wei 단위 출력

> web3.fromWei(eth.getBalace(eth.accounts[0]),"ether") // ether 단위 잔고 확인

**5) 송금 실행 ( 송금을 하기 위한 Transaction 의 발행은 수수료가 들어가기 때문에 항상 잠금 상태)**

➔ 송금하고자 하는 계정의 잠금 상태를 해제해야만 송금 가능

> personal.unlockAccount(eth.accounts[0],"비밀번호")
( 잠금 해제 유효시간은 기본적으로 300 초이며 세번째 인자를 통해 변경 가능!)

> eth.sendTransaction({from:eth.accounts[0], to:eth.accounts[1], value: web3.toWei(10,"ether")})
➔ eth.accounts[0] 에서 eth.accounts[1]로 송금 / 10 ether 송금

➔ 1 번인덱스 계정 확인해보면 0 으로 나옴 ( Transaction 을 발행해도, 블록체인의 블록 안에 Transaction 을 포함시키지 않으면 Transaction 의 내용이 실행되지 않는다

➔ 즉, 블록이 생성될 때 Transaction 이 블록에 포함되므로 다시 채굴을 진행!

> eth.pendingTransactions // 하나의 계류 중인 Transaction 정보 확인 ➔ blockNumber 가 null 이라는 것은 아직 Transaction 이 블록에 포함되지 않았다는 것 ➔ 계류중인 Transaction 이 없어질때 까지 채굴 진행

**6) Background Geth ( 항상 백그라운드에서 동작하며 채굴을 수행하도록 설정 )**

- nohup : 로그아웃 후에도 프로세스가 종료되지 않음. 중지하기 위해 kill 명령 사용
- –mine : 채굴을 활성화
- –rpc : HTTP-RPC 서버를 활성화
- –rpcaddr "0.0.0.0" : HTTP-RPC 서버에 접속할 IP 를 지정. "0.0.0.0" 값은 모든 IP 가 접속가능하게 함
- –rpcport 8545 : HTTP-RPC 서버가 요청을 받기 위해 사용하는 포트 지정
- –rpccorsdomain "" : 자신의 노드에 RPC 로 접속할 IP 를 지정. ""값은 모든 IP 가 접속 가능
- –rpcapi "admin, db, eth, debug, miner, net, shh, txpool, personal, web3":
  (RPC 에서 사용이 허가되는 기능 모듈의 지정)
- verbosity 6: 로그 수준(default=3), 0=slient, 1=error, 2=warn, 3=info, 4=core, 5=debug, 6=detail
- –unlock 0,1:
  (계정 0 과 계정 1 을 다음의 –password 옵션으로 풀어 놓는다.
  geth 에서 암호입력없이 해당 계정 접근가능)

- –password:
  (password 가 저장되어 있는 경로명 포함한 파일 이름)

- &: (geth 를 백그라운드에서 실행하라는 명령)

$ nohup geth —networkid 4649 —nodiscover —maxpeers 0 —datadir dev/BlockChain-Project/testnet —mine —minerthreads 1 —rpc 2>> dev/BlockChain-Project/testnet/geth.log &
$ jobs. **(만약 상태가 stopped 으로 되어 있다면 )**
$ bg 1(번호 입력) ➔ running 상태로 바뀌면 해결!

$ geth attach rpc:http://localhost:8545

$ ps  // 실행중인 프로세스 확인 //  ps aux l grep geth

$ kill -9 PID

**7) JSON-RPC ( geth 기동 시 HTTP-RPC  서버 활성화 해서 원격으로 명령 실행하기 위한 방법**

$ nohup geth —networkid 4649 —nodiscover —maxpeers 0 —datadir dev/BlockChain-Project/testnet —mine —minerthreads 1 —rpc —rpcaddr “0.0.0.0” —rpcport 8545 —rpccorsdomain “*” —rpcapi “admin,db,eth,debug,miner,net,ssh,txpool,personal,web3” 2 >> dev/BlockChain-Project/testnet/geth.log &

# 스마트 컨트랜트

1.

# Web3.js ( using node.js)

## 1. Web3 설치 및 연동

**$ npm install Ethereum/web3 –save**

**Web3 설치 및 연동**

- web3 모듈 설치 : npm install ethereum/web3 –save
- Localhost에 http로 연결

```
//---------------------
// Web3 연결
//---------------------
var Web3 = require('web3');
var web3 = new Web3(new Web3.providers.HttpProvider("http://127.0.0.1:8545"));
```

## 2. Ganache 실행 ( 가상 이더리움 클라이언트 )

**$ npm install -g ganache-cli  ( 설치 )**

**$ ganache-cli (실행)**

➔ 이더리움 애플리케이션을 빠르게 개발할 수 있도록 도와주는 development-tool

➔ 클라이언트를 직접 생성, 관리 할 필요 없이 명령어 한 줄로 테스트용 지갑 생성

➔ 스마트 컨트랙트와 서버 코드 작성에만 집중 가능하게 해줌