violas testnet

Part One: Run Violas Client Under Ubuntu

Run a CLI Client to Connect to Violas Network

- On Microsoft Windows 10 or up to connect to Violas client
- Open an Ubuntu terminal
- Download Violas Client: `wget https://violas.io/client/violas-client.tar.gz`
- Configure Ubuntu terminal to connect to Violas testnet
- Decompress the client tar file: `tar -zxvf violas-client.tar.gz && cd violas-client/Ubuntu`
- Connect to Violas testnet: `./start_cli.sh`
- On Mac to connect to Violas testnet
- Decompress the client tar file and change to where the client is installed: `tar -zxvf violas-client.tar.gz && cd violas-client/MacOS10.14`
- Connect to Violas testnet: `./start_cli.sh`

If successfully connected to the Violas testnet:

```
root@DESKTOP-S7H2Q83:~/violas_client# ./violas_client -a ac.testnet.violas.io -p 40001 -s $(pwd)/violas_consensus_peers.config.toml -m $(pwd)/faucet
Connected to validator at: ac.testnet.violas.io:40001
usage: <command> <args>

Use the following commands:

account | a
        Account operations
query | q
        Query operations
transfer | transferb | t | tb
        <sender_account_address>|<sender_account_ref_id> <receiver_account_address>|<receiver_account_ref_id> <number_of_coins> [gas_unit_price_in_m
ult=0)] [max_gas_amount_in_micro_libras (default 140000)] Suffix 'b' is for blocking.
        Transfer coins (in libra) from account to another.
dev
        Local move development
help | h
        Prints this help
quit | q!
        Exit this client


Please, input commands:

violas%
```

Due to network instability, it might cause the below error. Please try to connect to Violas testnet again

```
Not able to connect to validator at ac.testnet.violas.io:40001, error RpcFailure(RpcStatus { status: RpcStatusCode(4), details:
Some("Deadline Exceeded") })
```

On Windows 10 install Ubuntu for Windows，uses the same way on Ubuntu to connect to the Violas testnet.

Run a CLI Client to Connect to Libra Network

- Connect to Libra testnet:
- Download the Violas client from a Ubuntu terminal prompt: `wget wget https://violas.io/client/violas-client.tar.gz`
- On Ubuntu to connect the Libra testnet
- Decompress the tar file and change to where the client is installed: `tar -zxvf violas-client.tar.gz && cd violas-client/Ubuntu`
- Connect to the Libra testnet: `./violas-client -a ac.testnet.libra.org -p 8000 -s $(dirname "$PWD")/libra_consensus_peers.config.toml`
- On Mac to connect to the Libra testnet
- Decompress the tar file and change to where the client is installed: `tar -zxvf violas-client.tar.gz && cd violas-client/MacOS10.14`
- Connect to the Libra testnet: `./violas-client -a ac.testnet.libra.org -p 8000 -s $(dirname "$PWD")/libra_consensus_peers.config.toml`

If successfully connected to the Libra testnet:

```
root@DESKTOP-S7H2Q83:~/violas_client# ./violas_client -a ac.testnet.libra.org -p 8000 -s $(pwd)/libra_consensus_peers.config.toml
Connected to validator at: ac.testnet.libra.org:8000
usage: <command> <args>

Use the following commands:

account | a
        Account operations
query | q
        Query operations
transfer | transferb | t | tb
        <sender_account_address>|<sender_account_ref_id> <receiver_account_address>|<receiver_account_ref_id> <number_of_coins> [gas_unit_price_in_m
ult=0)] [max_gas_amount_in_micro_libras (default 140000)] Suffix 'b' is for blocking.
        Transfer coins (in libra) from account to another.
help | h
        Prints this help
quit | q!
        Exit this client


Please, input commands:

violas%
```

Part Two: First Transaction on the Violas Testnet

First Transaction on the Violas Testnet

- First Transaction on the Violas Testnet
- From violas% prompt run the command to create the account A： account create. In order to simulate transfer functionality between two accounts, please create another account B. The account A and B are shown below：

```
violas% account create
>> Creating/retrieving next account from wallet
Created/retrieved account #0 address 2d935053c46db9b24af1be38520048f93fbc6742ce7dab4b609af2a64a0ea857
violas% account create
>> Creating/retrieving next account from wallet
Created/retrieved account #1 address 208726d49ae923ee77c22f05b142289fdd0e4cf800fa92fa20c9f3b1666b7101
violas%
```

- List the account sequence number： account list

```
violas% account list
User account index: 0, address: 2d935053c46db9b24af1be38520048f93fbc6742ce7dab4b609af2a64a0ea857, sequence number: 0, status: Local
User account index: 1, address: 208726d49ae923ee77c22f05b142289fdd0e4cf800fa92fa20c9f3b1666b7101, sequence number: 0, status: Local
Faucet account address: 000000000000000000000000000000000000000000000000000000000a550c18, sequence_number: 76, status: Persisted
violas%
```

- Query account balance： query balance 0. 0 indicates the first account address ID. It can also query the account using its address： `query balance 2d935053c46db9b24af1be38520048f93fbc6742ce7dab4b609af2a64a0ea857` ：

```
violas% query balance 0
Balance is: 10000.000000
violas% query balance 1
Balance is: 0.000000
violas%
```

- Mint 10000 coin： `account mint 0 10000` . 0 indicates the first account address ID but can use the actual address as well：

```
violas% account mint 0 10000
>> Minting coins
Mint request submitted
```

- Transfer 500 coin from the account A to the account B： `transfer 0 1 500` .Where 0 and 1 are the indicators of the account A and B but can also use their actual addresses as well：

```
violas% transfer 0 1 500
>> Transferring
Transaction submitted to validator
To query for transaction status, run: query txn_acc_seq 0 0 <fetch_events=true|false>
violas% query balance 0
Balance is: 9500.000000
violas% query balance 1
Balance is: 500.000000
violas%
```

- Query the transfer status： `query txn_acc_seq 0 0 true`：

```
violas% query txn_acc_seq 0 0 true
>> Getting committed transaction by account and sequence number
Committed transaction: SignedTransaction {
 raw_txn: RawTransaction {
        sender: 2d935053c46db9b24af1be38520048f93fbc6742ce7dab4b609af2a64a0ea857,
        sequence_number: 0,
        payload: {,
                transaction: peer_to_peer_transaction,
                args: [
                        {ADDRESS: 208726d49ae923ee77c22f05b142289fdd0e4cf800fa92fa20c9f3b1666b7101},
                        {U64: 500000000},
                ]
        },
        max_gas_amount: 140000,
        gas_unit_price: 0,
        expiration_time: 1576482805s,
},
 public_key: Ed25519PublicKey(
    PublicKey(CompressedEdwardsY: [103, 163, 246, 99, 180, 198, 203, 106, 87, 109, 121, 86, 193, 50, 34, 178, 44,
, 82, 177, 33]), EdwardsPoint{
        X: FieldElement51([1692053173970412, 163370015240721, 449094812976280, 2120939900703331, 163275907784489])
        Y: FieldElement51([1062903003652967, 47069312773465, 1278374245337291, 1515794210230601, 592727504056808])
        Z: FieldElement51([1, 0, 0, 0, 0]),
        T: FieldElement51([1197108655461124, 1817760676977501, 1815421710891479, 1386823326980791, 143414923751509
    }),
),
 signature: Ed25519Signature(
    Signature( R: CompressedEdwardsY: [68, 5, 196, 121, 79, 229, 201, 172, 207, 252, 22, 140, 121, 24, 17, 229, 24
8, 141, 12, 150, 153], s: Scalar{
        bytes: [104, 178, 52, 111, 103, 225, 96, 159, 3, 82, 12, 149, 147, 51, 44, 152, 201, 106, 75, 167, 191, 19
    } ),
),
 }
Events:
ContractEvent { key: 0100000000000002d935053c46db9b24af1be38520048f93fbc6742ce7dab4b609af2a64a0ea857, index: 0, t
00000000000000000000000000000000000000000000000000, module: Identifier("LibraAccount"), name: Identifier("SentPaymentE
t { amount: 500000000, account: 208726d49ae923ee77c22f05b142289fdd0e4cf800fa92fa20c9f3b1666b7101, metadata: [] } }
ContractEvent { key: 0000000000000000208726d49ae923ee77c22f05b142289fdd0e4cf800fa92fa20c9f3b1666b7101, index: 0, t
00000000000000000000000000000000000000000000000000, module: Identifier("LibraAccount"), name: Identifier("ReceivedPaym
Event { amount: 500000000, account: 2d935053c46db9b24af1be38520048f93fbc6742ce7dab4b609af2a64a0ea857, metadata: []
violas%
```

Part Three: Deploy Move Contract

Compile and Publish Move Modules

Compile and Publish Vtoken Move Contract

- Violas Testnet provides dev command to compile and publish Move contract. Here is the steps：
- Download and compile Violas Testnet client then execute the following commands in order：

```
git clone https://github.com/palliums-developers/Violas.git
cd Violas && git checkout testnet
./scripts/dev_setup.sh
./scripts/cli/start_cli_testnet.sh
```

- Create and publish account of Move contract： `account create`
- Mint coin: `account mint 0 10000`
- Query account balance： `query balance 0`
- Record the account address which will be used in the Move contract as the Move contract publish address：

```
violas% account create
>> Creating/retrieving next account from wallet
Created/retrieved account #0 address ff78ef18a5467a829a7a81dda702d12c05cf5fca5440764d5f72154f6cb07185
violas% account mint 0 1000
>> Minting coins
Mint request submitted
violas% query balance 0
Balance is: 1000.000000
violas%
```

- Create Move Module and use the above account address to replace the address in the Move contract. Make sure the 0x must be preserved Vtoken.mvir example：

```
module ViolasToken {

import 0x00.LibraAccount;
import 0x0.LibraCoin;
import 0x00.Hash;
import 0x0.U64Util;
```

```
import 0x0.AddressUtil;
import 0x0.BytearrayUtil;
import 0x0.Vector;
resource Owner {}

// A resource representing the ViolasToken
resource T {
    value: u64,
}

resource Info {
    magic: u64,
    token: address,
    allinone_events: LibraAccount.EventHandle<Self.AllInOneEvent>,
}

resource OwnerData {
    data: bytearray,
    owner: address,
    bulletins: Vector.T<bytearray>,
}

struct AllInOneEvent {
    etype: u64,
    sender: address,
    receiver: address,
    token: address,
    amount: u64,
    price: u64,
    data: bytearray,
}

// Publishes an initial zero dToken to the sender.
public publish() acquires Info {

    let sender: address;

    sender = get_txn_sender();

    if(exists<T>(copy(sender))) {
        return;
    }

    if (copy(sender) == 0xff78ef18a5467a829a7a81dda702d12c05cf5fca5440764d5f72154f6cb07185) {
        move_to_sender<Owner>(Owner{});
        move_to_sender<OwnerData>(OwnerData{ data: h"", owner:
0xff78ef18a5467a829a7a81dda702d12c05cf5fca5440764d5f72154f6cb07185, bulletins: Vector.empty<bytearray>() });
    }

    move_to_sender<T>(T{ value: 0 });

    move_to_sender<Info>(Info{
        magic: 123456789,
        token: 0xff78ef18a5467a829a7a81dda702d12c05cf5fca5440764d5f72154f6cb07185,
        allinone_events: LibraAccount.new_event_handle<Self.AllInOneEvent>(),
    });

    Self.emit_events(0, copy(sender), 0, 0, h"");

    return;
}

require_published() {
    let sender: address;
    let is_present: bool;
    sender = get_txn_sender();
    is_present = exists<T>(move(sender));
    assert(move(is_present), 101);
    return;
}

require_owner() {
    let sender: address;
    let is_present: bool;
    sender = get_txn_sender();
    is_present = exists<Owner>(move(sender));
    assert(move(is_present), 102);
    return;
```

```
}

// Mint new dTokens.
mint(value: u64): Self.T {
    Self.require_published();
    Self.require_owner();
    return T{value: move(value)};
}

public mint_to_address(payee: address, amount: u64) acquires T, Info {

    let token: Self.T;
    token = Self.mint(copy(amount));

    // Mint and deposit the coin
    Self.deposit(copy(payee), move(token));

    Self.emit_events(1, copy(payee), copy(amount), 0, h"");

    return;
}

public mint_to_address_with_data(payee: address, amount: u64, data: bytearray) acquires T, Info {

    let token: Self.T;
    token = Self.mint(copy(amount));

    // Mint and deposit the coin
    Self.deposit(copy(payee), move(token));

    Self.emit_events(13, copy(payee), copy(amount), 0, move(data));

    return;
}

public zero(): Self.T {
    return T{ value: 0 };
}

value(coin_ref: &Self.T): u64 {
    return *&move(coin_ref).value;
}

// Returns an account's dToken balance.
balance(): u64  acquires T {
    let sender: address;
    let token_ref: &Self.T;
    let token_value: u64;

    Self.require_published();

    sender = get_txn_sender();
    token_ref = borrow_global<T>(move(sender));
    token_value = *(&move(token_ref).value);

    return move(token_value);
}

// Deposit owned tokens to a payee's address
deposit(payee: address, to_deposit: Self.T) acquires T {
    let sender: address;
    let amount: u64;

    let payee_token_ref: &mut Self.T;
    let payee_token_value: u64;
    let to_deposit_value: u64;

    Self.require_published();

    sender = get_txn_sender();

    payee_token_ref = borrow_global_mut<T>(copy(payee));
    payee_token_value = *(&copy(payee_token_ref).value);

    // Unpack and destroy to_deposit tokens
    T{ value: to_deposit_value } = move(to_deposit);

    amount = copy(to_deposit_value);
```

```
    // Increase the payees balance with the destroyed token amount
    *(&mut move(payee_token_ref).value) = move(payee_token_value) + move(to_deposit_value);

    return;
}

// Withdraw an amount of tokens of the sender and return it.
withdraw(amount: u64): Self.T acquires T {
    let sender: address;
    let sender_token_ref: &mut Self.T;
    let value: u64;

    Self.require_published();

    sender = get_txn_sender();

    sender_token_ref = borrow_global_mut<T>(move(sender));
    value = *(&copy(sender_token_ref).value);

    // Make sure that sender has enough tokens
    assert(copy(value) >= copy(amount), 103);

    // Split the senders token and return the amount specified
    *(&mut move(sender_token_ref).value) = move(value) - copy(amount);
    return T{ value: move(amount) };
}

pay_from_sender(payee: address, amount: u64) acquires T {
    let sender: address;
    let to_pay: Self.T;

    Self.require_published();

    sender = get_txn_sender();

    to_pay = Self.withdraw(copy(amount));
    Self.deposit(copy(payee), move(to_pay));

    return;
}

public transfer(payee: address, amount: u64) acquires T, Info {
    Self.pay_from_sender(copy(payee), copy(amount));
    Self.emit_events(2, copy(payee), copy(amount), 0, h"");
    return;
}

public transfer_with_data(payee: address, amount: u64, data: bytearray) acquires T, Info {
    Self.pay_from_sender(copy(payee), copy(amount));
    Self.emit_events(12, copy(payee), copy(amount), 0, move(data));
    return;
}

public transfer_vcoin_with_data(payee: address, amount: u64, data: bytearray) acquires Info {
    LibraAccount.pay_from_sender(copy(payee), copy(amount));
    Self.emit_events(10, copy(payee), copy(amount), 0, move(data));
    return;
}

public record(data: bytearray) acquires Info {
    let sender: address;
    sender = get_txn_sender();
    Self.emit_events(11, copy(sender), 0, 0, move(data));
    return;
}

emit_events(etype: u64, receiver: address, amount: u64, price: u64, data: bytearray) acquires Info {

    let sender: address;
    let token: address;

    let allinone_event: Self.AllInOneEvent;

    let sender_info_ref: &mut Self.Info;
    let receiver_info_ref: &mut Self.Info;

    sender = get_txn_sender();
```

```
        token = 0xff78ef18a5467a829a7a81dda702d12c05cf5fca5440764d5f72154f6cb07185;

        allinone_event = AllInOneEvent {
            etype: copy(etype),
            sender: copy(sender),
            receiver: copy(receiver),
            token: copy(token),
            amount: move(amount),
            price: move(price),
            data: move(data),
        };

        sender_info_ref = borrow_global_mut<Info>(copy(sender));
        LibraAccount.emit_event<Self.AllInOneEvent>(&mut move(sender_info_ref).allinone_events, copy(allinone_event));

        if((copy(etype) == 1)   ||
           (copy(etype) == 2)   ||
           (copy(etype) == 5)   ||
           (copy(etype) == 8)   ||
           (copy(etype) == 10)  ||
           (copy(etype) == 12)  ||
           (copy(etype) == 13)  ||
           (copy(etype) == 14)  ||
           (copy(etype) == 15)  ||
           (copy(etype) == 16))
        {
            receiver_info_ref = borrow_global_mut<Info>(copy(receiver));
            LibraAccount.emit_event<Self.AllInOneEvent>(&mut move(receiver_info_ref).allinone_events, copy(allinone_event));
        }

        return;
        }
    }
```

- Compile Move Module To compile Vtoken.mvir, use the dev compile command： `dev compile 0 /root/violas-client/move/Vtoken.mvir module` :

```
violas% dev compile 0 /root/violas-client/move/Vtoken.mvir module
>> Compiling program
    Finished dev [unoptimized + debuginfo] target(s) in 1.65s
     Running `/root/Violas/target/debug/compiler -l /tmp/61e106c36cabc0067506d30fe63a4761.mvir -m`
    Finished dev [unoptimized + debuginfo] target(s) in 1.45s
     Running `/root/Violas/target/debug/compiler /tmp/61e106c36cabc0067506d30fe63a4761.mvir -a ff78ef18a5467a829a7a81dda702d12c05cf5fca5440764
Successfully compiled a program at /tmp/61e106c36cabc0067506d30fe63a4761.mv
violas%
```

The generated bycode is stored to the default path tmp as shown above in red color. In the below publish command will use this *.mv file

- Publish Compiled Module： `dev publish 0 /tmp/61e106c36cabc0067506d30fe63a4761.mv` :

```
violas% dev publish 0 /tmp/61e106c36cabc0067506d30fe63a4761.mv
waiting .transaction is stored!
no events emitted
Successfully published module
violas%
```

Compile and Execute Scripts

Compile and execute Mint scripts

1. Create Mint Script：mint.mvir script. Replace the address in the script by the address to publish the Move contract then mint coin to the address.

mint.mvir example：

```
import 0xff78ef18a5467a829a7a81dda702d12c05cf5fca5440764d5f72154f6cb07185.ViolasToken;

main(payee: address, amount: u64) {
    ViolasToken.mint_to_address(move(payee), move   (amount));

    return;
}
```

1. Compile Mint Script To compile your mint script, use the dev compile command： `dev compile 0 /root/violas-client/move/mint.mvir script` Note：When compile Module and Script，the last parameter indicates it is Module or Script：

```
violas% dev compile 0 /root/violas-client/move/mint.mvir script
>> Compiling program
    Finished dev [unoptimized + debuginfo] target(s) in 1.44s
     Running `/root/Violas/target/debug/compiler -l /tmp/15940639f4f204b979587ca240f76fa4.mvir`
    Finished dev [unoptimized + debuginfo] target(s) in 1.44s
     Running `/root/Violas/target/debug/compiler /tmp/15940639f4f204b979587ca240f76fa4.mvir -a ff78ef18a5467a829a7a81dda702d12c05cf5fca5440764
843f87e1111b8bba43a6bf020a01532`
Successfully compiled a program at /tmp/15940639f4f204b979587ca240f76fa4.mv
```

The generated bycode is stored to the default path tmp as shown above in red color. In the below publish command will use this *.mv file. The below Execute Mint Script will use this* .mv file

2. Execute Mint Script with two parameters address and amount： `dev execute 0 /tmp/15940639f4f204b979587ca240f76fa4.mv 0xff78ef18a5467a829a7a81dda702d12c05cf5fca5440764d5f72154f6cb07185 100000` .

Replace the address in the command to the address to be mint to but keep the 0x. 100000 is the mint amount:

```
violas% dev execute 0 /tmp/15940639f4f204b979587ca240f76fa4.mv  0xff78ef18a5467a829a7a81dda702d12c05cf5fca5440764d5f72154f6cb07185 100000
waiting transaction is stored!
no events emitted
Successfully finished execution
```

Compile and Execute Transaction Script

1. Create Transaction Script: transfer.mvir script and replace the address by the newly created address.

transfer.mvir example:

```
import 0xff78ef18a5467a829a7a81dda702d12c05cf5fca5440764d5f72154f6cb07185.ViolasToken;

main(payee: address, amount: u64) {
    ViolasToken.transfer(move(payee), move(amount));
    return;
}
```

1. Compile Transaction Script To compile your transaction script, use the dev compile command: `dev compile 0 /root/violas-client/move/transfer.mvir script` Note: When compile Module and Script, the last parameter is the indicator of Module or Script:

```
violas% dev compile 0 /root/violas-client/move/transfer.mvir script
>> Compiling program
    Finished dev [unoptimized + debuginfo] target(s) in 1.47s
     Running `/root/Violas/target/debug/compiler -l /tmp/20f5d4bea66ba7f659fea3db74b78ee6.mvir`
    Finished dev [unoptimized + debuginfo] target(s) in 1.47s
     Running `/root/Violas/target/debug/compiler /tmp/20f5d4bea66ba7f659fea3db74b78ee6.mvir -a ff78ef18a5467a829a7a81dda702d12c05cf5fca5440764
fbe0a4d1d142c2872e725670db6a670`
Successfully compiled a program at /tmp/20f5d4bea66ba7f659fea3db74b78ee6.mv
violas%
```

The generated bycode is stored to the default path tmp as shown above in red color. In the below publish command will use this *.mv file. The below Execute Transaction Script will use this* .mv file.

2. Execute Transaction Script to create a new address to receive the coin/token with two parameters address to transfer coin and amount of coin to transfer: `dev execute 0 /tmp/20f5d4bea66ba7f659fea3db74b78ee6.mv 0x07f64b38955886f0539733a69fcb8b124d32188aaa1771cfca90b6837694a11e 500` . Replace the above address to your receiving address of coin/token. The 0x is required for the address:

```
violas% account list
User account index: 0, address: ff78ef18a5467a829a7a81dda702d12c05cf5fca5440764d5f72154f6cb07185, sequence number: 2, status: Persisted
User account index: 1, address: 07f64b38955886f0539733a69fcb8b124d32188aaa1771cfca90b6837694a11e, sequence number: 0, status: Unknown
Faucet account address: 000000000000000000000000000000000000000000000000000000000a550c18, sequence_number: 15, status: Persisted
violas% dev execute 0 /tmp/20f5d4bea66ba7f659fea3db74b78ee6.mv  0x07f64b38955886f0539733a69fcb8b124d32188aaa1771cfca90b6837694a11e 500
waiting ...transaction is stored!
no events emitted
Successfully finished execution
violas%
```

Above are the work flow to issue Vtoken, Mint, Transaction.

Part Four Common Commands

Common Commands

Use help to lookup commands and how to use them. For example: `dev --help`

- Violas client connects to testnet

```
USAGE:
    violas_client [FLAGS] [OPTIONS] --host <host> --validator-set-file <validator-set-file>

FLAGS:
    -h, --help       Prints help information
    -r, --sync       If set, client will sync with validator during wallet recovery.
    -V, --version    Prints version information
    -v, --verbose    Verbose output.

OPTIONS:
    -m, --faucet-key-file-path <faucet-account-file>
            Path to the generated keypair for the faucet account. The faucet account can be used to mint coins. If not
            passed, a new keypair will be generated for you and placed in a temporary directory. To manually generate a
            keypair, use generate-keypair: `cargo run -p generate-keypair -- -o <output_file_path>`
    -f, --faucet-server <faucet-server>
            Host that operates a faucet service If not passed, will be derived from host parameter

    -a, --host <host>                                   Host address/name to connect to.
    -n, --mnemonic-file <mnemonic-file>
            File location from which to load mnemonic word for user account address/key generation. If not passed, a new
            mnemonic file will be generated by libra-wallet in the current directory.
    -p, --port <port>                                   Admission Control port to connect to. [default: 8000]
    -s, --validator-set-file <validator-set-file>
            File location from which to load config of trusted validators. It is used to verify validator signatures in
            validator query response. The file should at least include public key of all validators trusted by the
            client - which should typically be all validators on the network. To connect to testnet, use
```

```
            'libra/scripts/cli/consensus_peers.config.toml'. Can be generated by config-builder for local testing:
            `cargo run --bin config-builder` But the preferred method is to simply use libra-swarm to run local networks
```

- Account command parameters:

```
usage: account <arg>

Use the following args for this command:

create | c
    Create an account. Returns reference ID to use in other operations
list | la
    Print all accounts that were created or loaded
recover | r <file_path>
    Recover Libra wallet from the file path
write | w <file_path>
    Save Libra wallet mnemonic recovery seed to disk
mint | mintb | m | mb <receiver_account_ref_id>|<receiver_account_address> <number_of_coins>
    Mint coins to the account. Suffix 'b' is for blocking
```

- Query command parameters:

```
usage: query <arg>

Use the following args for this command:

balance | b <account_ref_id>|<account_address>
    Get the current balance of an account
sequence | s <account_ref_id>|<account_address> [reset_sequence_number=true|false]
    Get the current sequence number for an account, and reset current sequence number in CLI (optional, default is false)
account_state | as <account_ref_id>|<account_address>
    Get the latest state for an account
txn_acc_seq | ts <account_ref_id>|<account_address> <sequence_number> <fetch_events=true|false>
    Get the committed transaction by account and sequence number.  Optionally also fetch events emitted by this transaction.
txn_range | tr <start_version> <limit> <fetch_events=true|false>
    Get the committed transactions by version range. Optionally also fetch events emitted by these transactions.
event | ev <account_ref_id>|<account_address> <sent|received> <start_sequence_number> <ascending=true|false> <limit>
    Get events by account and event type (sent|received).
```

- Transfer command parameters:

```
transfer | transferb | t | tb
    <sender_account_address>|<sender_account_ref_id> <receiver_account_address>|<receiver_account_ref_id> <number_of_coins>
[gas_unit_price_in_micro_libras (default=0)] [max_gas_amount_in_micro_libras (default 140000)] Suffix 'b' is for blocking.
```

- Dev command parameters:

```
usage: dev <arg>

Use the following args for this command:

compile | c <sender_account_address>|<sender_account_ref_id> <file_path> <module|script> [output_file_path (compile into tmp file
by default)]
    Compile move program
publish | p <sender_account_address>|<sender_account_ref_id> <compiled_module_path>
    Publish move module on-chain
execute | e <sender_account_address>|<sender_account_ref_id> <compiled_module_path> [parameters]
    Execute custom move script
```