Commands Manual

INDEX

1. PREFACE

Aladin is the revolutionary DApp store which will make an impression on the DApp developers with its easier functionalities. Aladin is a Blockchain Platform for financially rewarding businesses and community. It is natively first dapp store where DApp developers will be rewarded. Aladin has a unique token economics model which will ensure to reward all the stakeholders involved

Aladin is a network that will impact anything which needs to operate in a decentralized environment. It is a new approach to how applications are managed over the internet. Aladin provides the capacities that applications use on the internet and brings them to a decentralized network

The Aladin network creates new Aladin tokens per year to reward block producers, DApp developers, and users who will vote for the Block Producers for showing their active participation in elections. During the first year, the network will feature 5% new coin inflation and this amount will decrease by 1% per year until it reaches 2%. Additionally, New inflated tokens would be divided into two equal parts. 50% of new inflated tokens would go to the network part and remaining would go to the infrastructure side. From the 50% of network ALADIN has distributed tokens among various parties. 60% would go to the top 21 Block producers, 15% would go to the top 50 witnesses, 25% would go to the DApp developers, 5% would go to the voters. In ALADIN network Voters will also be rewarded for the voters for showing their active participation in the elections. From the 50% of Infrastructure ALADIN has distributed tokens among the Community Development Team with 65% of tokens, Marketing Team with 10% of tokens and Founding Team with 25% of tokens.The start of the year would also see Block Producer also getting reward of 44% from the community development.

In The Aladin Network all the parties have to meet the prerequisites to become a member of the ALADIN Network.

2. Prerequisites

To prerequisites to use ALADIN Network are as follows:

- OS: Ubuntu 18.04
- RAM: 16GB - 1TB
- CPUs: 4vCPUs - 8vCPUs
- SSD: 30GB - 500GB

These are the minimum to average hardware prerequisites to setup the ALADIN network and use some set of commands of ALADIN.

One producing node and multiple fully redundant backups in a fail-over model comprise our core layer. The main producing node is the workhorse of the operation. A backup is activated during system upgrades. Following specification is for the Core Layer of ALADIN Network

- 80 GBs - 3 TBs of RAM (as needed)
- 4-8 vCPUs
- 1 TB of SSD Storage

The most important resource is RAM with a need of around 3TB for the OS, but most BP prefer Linux.

As ALADIN can be used by multiple parties, they have to meet hardware requirements and some software prerequisites as well. Block Producers, DApp Developers and Voters are the main parties of the ALADIN network, they all have to set up different kind of environment setup with the given prerequisites.

Following command manual presuming that all parties have met all types of hardware/software requirements with ALAIO, ALAIO.CDT, ALAIO.Contracts installed on their machines.
ALA: https://github.com/ALADINIO/ala.git
ALAIO.CDT: https://github.com/ALADINIO/alaio.cdt.git
ALAIO.Contracts: https://github.com/ALADINIO/alaio.contracts.git

a. **For Block Producer**

Block Producers are an essential part of the ALADIN network, They have to meet the hardware requirements to set up their nodes for the ALADIN network and meet some software requirements as well, Software requirements are as follows:

- alaio v1.7.0 deb package installed in their machine
- alaio.cdt v1.5.0 deb package installed in their machine
- alaio.contracts v1.6.0 deb package installed in their machine.
- Clang-4.0, lldb-4.0, libclang-4.0-dev, cmake, make, automake, libbz2-dev, libssl-dev, libgmp3-dev, autotools-dev, build-essential, libicu-dev, python2.7-dev, python3-dev, autoconf, libtool, curl, zlib1g-dev, doxygen,graphviz

ALADIN provides Contract Development Toolkit, If Block Producer has a fresh machine they would need to download the source code of the alaio.cdt package, build and install the package from the source so that they doesnt need to download all the required packages separately. ALADIN CDT package will download all the required packages.

To become a Block Producer on ALADIN Network, Block Producer needs to follow some steps to setup a node and started producing the blocks. Top 21 Block Producers can be elected by the voters and based on the number of votes, Block Producer is assigned the number between 1 to 21. ALADIN has top 50 Block Producers, so Top 21 will be producing the blocks and from 22nd to 50th producers will be known as Witnesses. ALADIN has reward distribution for the witnesses with at least 0.5% of votes. The reward would be distributed equally amongst those.

Token Economics model of ALADIN distributes 15% of witness rewards equally to the top 50 Block Producers, means irrespective of their position or votes top 50 will always get 15% of witness rewards equally, but they all have to at least 0.5% of votes in their account.

ALADIN has set some restrictions on the Block Producers as well for choosing the genuine Block Producers to run the network. Block Producers has to have at least 100’000 ALA staked tokens in their account then they will be eligible to become the Block Producers and Block Producer can not vote to other Block Producers.

The Payout period of Block Producers and witnesses would be once per day.

b. **For DApp Developers**

On ALADIN platform DApp developers from around the globe can come and host their dapps on the network which is based on Delegated Proof of Stake (DPoS) consensus. ALADIN Network categorize all the DApps in two categories, 1. Number of users onboard on the DApp, 2. The total monetary volume of incoming transactions on the DApp.

ALADIN provides the functionality of built in universal oracle for the DApp developers, It's all up to the DApp developer to make the DApp of their own choice. If Developer wants to use the feature of universal oracle then the requirements will be a little bit different.

→ **DApps which will be having the feature of oracle:**

DApp Developer will have to follow the standard provided by Aladin while making the smart contracts for their DApps as oracle will be taking some arguments from the smart contracts so ALADIN expect from the DApp developer to maintain the standard of the smart contracts.

→ **DApps which will not be having the feature of oracle:**

If DApp developer chooses to not use the functionality of oracle then DApp developer can write the smart contract as per their requirements.

Smart contracts are programmed using C++, which boosts programming flexibility. ALADIN smart contracts are implemented onto the blockchain in the form of pre-compiled Web Assembly (WASM), which promotes faster execution of contracts when compared to other platforms which supports smart contracts. WASM is compiled with C/C++ via means of clang and LLVM. DApp Developers have to have knowledge of C/C++ in order to be able to code smart contracts on ALADIN's blockchain. Even though C can be used to create contracts, it is highly recommended to use the ALAIO C++ API, which strengthens contract's safety and renders its code easily readable.

When DApp Developer will onboard themselves on ALADIN Platform, ALADIN will ask to submit the preference and based on their preference DApp Developer will be rewarded the ALA tokens at the end of the 30 days.

DApp Developers payout period would be once per month.

c. **For Voters**

ALADIN Network will always be looking for active participation of the users for the election of the Block Producer and this makes ALADIN different from other Blockchain Platforms, as ALADIN has different tokenomics model in which voters will be rewarded with 5% of tokens allocated to the Network.

On ALADIN Network the user would need to stake some coinns

Voters has to stake minimum amount of tokens to give the votes to the block producers, If user does not have the staked tokens in their account then he is not eligible for the voting as well as can not claim reward. If User has staked more than 1000 then ALADIN restricts user to vote not more than 1000 it means it does not matter if user has more than 1000 staked tokens in his account but at a time user can vote only 1000 that makes more participation of the voters on the election and on the basis of the number of votes a user has voted to the block producers, user will be rewarded ALA token at the end of the 30 days.

For voting user must have to meet the bare minimum hardware requirements and installed the required version of **alacli** on his node/machine. Users/Voters can communicate with any API endpoint provided by the Block Producers.

3. Block Producer Environment Setup

All Block producers must have common genesis file which indicates the information of the genesis node and the public key of genesis node of aladin.
Block Producer has to build and install the ALAIO, ALAIO.CDT and ALAIO.Contracts package. Block Producer will receive their mnemonic code in their email when they register themselves on ALADIN Platform. They will have to execute a command to receive the private key and import that into wallet.

→ **For generating a Private key from mnemonic code**

curl -X POST "https://api.aladinnetwork.org/users/getKeys" -H "accept: multipart/form-data" -H "Content-Type: application/x-www-form-urlencoded" -d "mnemonicCode=**MNEMONIC_CODE_RECIEVED_IN_EMAIL**&password=**PASSWORD_WHEN_YOU_REGISTER**&accountNo=1"

This Command has a parameter **“accountNo”** by default it would be 1, ALADIN facilitate the BP to create more than one ID with the same mnemonic code so if BP wants to create another ID then parameter **“accountNo”** would be 2, and BP will get the different set of Public and Private key.

→ **For creating a wallet**

./bin/alacli wallet create

When this command executes successfully it will give the password of the wallet, Block Producer has to store the password in a safe place so whenever their wallet is locked BP can unlock their wallet with the help of this password.By delinquency “default” wallet is created but BP can create the wallet with specific name.

./bin/alacli create wallet **--name=YOUR_WALLET_NAME** --to-console

With the “**--name**” parameter block producer can create the wallet with specific name.

→ **For unlocking a wallet**

```
./bin/alacli wallet unlock
```

When this command executes it will ask for the wallet password, Block producer has to enter their password and then wallet will be unlocked.
If Block Producer has set up their wallet with the specific name then there will be a little bit difference in unlock command

```
./bin/alacli wallet unlock --name=YOUR_WALLET_NAME
```

→ **For creating a pair of Public and Private key**

```
./bin/alacli create key --to-console
```

When this command executes it will give you the Public and Private key. The prefix of Public key would be **ALA.** Once Block Producer has generated the pair of keys then BP has to import their private key into their wallet.

→ **For importing private key into the Wallet**

```
./bin/alacli wallet import
```

It will ask for the private key, Block Producer has to enter the private key in the console and once done, Block producer can see his public key as output. For cross verification

```
./bin/alacli wallet keys list
```

It will display all the public key which is present in the wallet. Block Producer can find the relative public key. If Block Producer has a wallet with specific name then the command would be a little bit different.

```
./bin/alacli wallet import --name=YOUR_WALLET_NAME
```

→ **genesis.json**

```
{
        "initial_timestamp":"2019-10-28T09:55:11.000",
        "initial_key": "GENESIS_NODE_PUB_KEY",
        "initial_configuration": {
                "max_block_net_usage": 1048576,
                "target_block_net_usage_pct": 1000,
                "max_transaction_net_usage": 524288,
                "base_per_transaction_net_usage": 12,
                "net_usage_leeway": 500,
                "context_free_discount_net_usage_num": 20,
                "context_free_discount_net_usage_den": 100,
                "max_block_cpu_usage": 100000,
                "target_block_cpu_usage_pct": 500,
                "max_transaction_cpu_usage": 50000,
                "min_transaction_cpu_usage": 100,
                "max_transaction_lifetime": 3600,
                "deferred_trx_expiration_window": 600,
                "max_transaction_delay": 3888000,
                "max_inline_action_size": 4096,
                "max_inline_action_depth": 4,
                "max_authority_depth": 6
        }
}
```

Every Block Producer has to create their own node with required plugins enabled and pass their pair of public key and private key, their producer name and all the p2p-peer-addresses with whom they want to connect. Generally, top 21 will connect with each other to sync the blockchain.
Block producer needs to create a wallet for storing the private keys and they will have to create their own public private key pair and store them into a safe place

→ **block-producer's genesis_script file:**

```
#!/bin/bash
DATADIR="./blockchain"
if [ ! -d $DATADIR ]; then
        mkdir -p $DATADIR;
fi
 alanode \
        --genesis-json $DATADIR"GENESIS.JSON_PATH" \
        --signature-provider BP_PUB_KEY=KEY:BP_PRIV_KEY \
        --plugin alaio::producer_plugin \
        --plugin alaio::chain_plugin \
        --plugin alaio::chain_api_plugin \
        --plugin alaio::history_plugin \
        --data-dir $DATADIR"/data" \
        --blocks-dir $DATADIR"/blocks" \
        --config-dir $DATADIR"/config" \
        --producer-name PRODUCER_NAME \
        --http-server-address 0.0.0.0:8888 \
        --p2p-listen-endpoint 0.0.0.0:4444 \
        --access-control-allow-origin=* \
        --contracts-console true \
        --http-validate-host=false \
        --verbose-http-errors \
        --chain-state-db-size-mb 8192 \
        --enable-stale-production \
        --p2p-peer-address OTHER_BP_ENDPOINT:4444 \
        --p2p-peer-address OTHER_BP_ENDPOINT:4444 \
        >> $DATADIR"/alanode.log" 2>&1 & \
        echo $! > $DATADIR"/alad.pid"
```

Block Producer can make the script and copy above contents in the script and run the script, If Block Producer wants to stop producing blocks for sometime then he should stop the alanode process. Then he can again start producing the blocks again for that he needs to start his node from the last when he has stopped their node and again sync the blockchain again.

→ **Block-producer's stop_script file:**

```
#!/bin/bash
DATADIR="./blockchain/"
if [ -f $DATADIR"/alad.pid" ]; then
        pid=`cat $DATADIR"/alad.pid"`
        echo $pid
        kill $pid
        rm -r $DATADIR"/alad.pid"
        echo -ne "Stoping Node"
        while true; do
                [ ! -d "/proc/$pid/fd" ] && break
                echo -ne "."
                sleep 1
        done
        echo -ne "\rNode Stopped. \n"
fi
```

→ **Block-producer's start_script file:**

This script would be a replica of the **genesis_script file,** Block Producer needs to remove the line → --genesis-json $DATADIR"**GENESIS.JSON_PATH**" \ from the **genesis_script file,** rest all content would remain the same.

The start script would help the Block Producers from start the chain again from where BP has stopped his node last time.

Once done, Block Producer will see in the log file that all blocks will be started fetching now Block Producer needs to register themselves on ALADIN-Network

→ **For registering Block Producers on ALADIN Platform**

./bin/alacli -u **API_ENDPOINT_OF_BP** system regproducer **BP_ACCOUNT_NAME BP_PUBLIC_KEY BP_WEBSITE BP_COUNTRY_CODE**

Block Producer has to follow the country code according to **ISO 3166** format. Block Producer can find their country code in following link: http://kirste.userpage.fu-berlin.de/diverse/doc/ISO_3166.html

→ **Block Producer can claim their command with the following command**

./bin/alacli -u **API_ENDPOINT_OF_BP** system claimrewards **BP_ACCOUNT_NAME**

ALADIN Network facilitates the Block Producers to claim different kinds of rewards as Universal Oracle is one of the most powerful features of ALADIN Platform so all the Block Producers has to set up the oracle on their node.

The payout period would be once per day so all Block Producers can claim their rewards once per day. Block Producers will receive the bpay (block rewards), wpay (witness rewards), opay (oracle rewards).

Oracle Rewards is distributed on the basis of successful request each BP makes and a penalty is levied on the Block Producer on the account of missing the request.
There could be four possible use cases

1. Oracle would rewarded on the basis of successful request each Block Producer and that would be independent of its block provider provided there isn't any failed request by that BP
2. Penalty would be deducted from the Oracle Reward , for e.g if the Oracle Reward is 10 ALA and penalty imposed on him is 5 ALA hence the net Oracle reward for that BP would be 5 ALA.
3. Penalty would be deducted from the Block reward, if it is greater than the Oracle Reward, for e.g if the penalty is 10 ALA and the Oracle Reward is 5 ALA , the remaining 5 ALA would be deducted from its Block Producer Reward.
4. In a case where the Oracle Penalty is greater then Block and Oracle Reward, the net reward would be negative and would be deducted whenever the Block Producer claims the reward next time

4. DApp Developer Environment Setup

DApp Developers must be familiar with the ALADIN's Command Line Arguments. DApp developer will receive their Public and Private key in their email when they register themselves on ALADIN Platform. They will have to execute a command to receive the private key and import that into wallet.

DApp Developer has to build and install ALAIO, ALAIO.CDT and ALAIO.Contracts packages to their node.

→ **For generating a Private key from mnemonic code**

curl -X POST "https://api.aladinnetwork.org/users/getKeys" -H "accept: multipart/form-data" -H "Content-Type: application/x-www-form-urlencoded" -d "mnemonicCode=**MNEMONIC_CODE_RECIEVED_IN_EMAIL**&password=**PASSWORD_WHEN_YOU_REGISTER**&accountNo=1"

This Command has a parameter **"accountNo"** by default it would be 1, ALADIN facilitate the Developer to create more than one ID with the same mnemonic code so if developer wants to create another ID then parameter **"accountNo"** would be 2, and developer will get the different set of Public and Private key.

When this command executes successfully it will give you a private key. They need to create a wallet for storing the private keys and they will have to create their own public private key pair and store them into a safe place.

→ **For creating a wallet**

./bin/alacli create wallet --to-console

When this command executes successfully it will give the password of the wallet, Developer has to store the password in a safe place so whenever their wallet is locked they can unlock their wallet with the help of this password. By delinquency "default" wallet is created but dev. can create the wallet with specific name.

./bin/alacli create wallet **--name=YOUR_WALLET_NAME** --to-console

With the "**--name**" parameter developer can create the wallet with specific name.

→ **For unlocking a wallet**

```
./bin/alacli wallet unlock
```

When this command executes it will ask for the wallet password, Developer has to enter their password and then wallet will be unlocked.
If Developer has set up their wallet with the specific name then there will be a little bit difference in unlock command.

```
./bin/alacli wallet unlock --name=YOUR_WALLET_NAME
```

→ **For importing private key into the Wallet**

```
./bin/alacli wallet import
```

It will ask for the private key, developer has to enter the private key in the console which they have generated from the first step and once done, Developer can see his public key as output.
For cross verification

```
./bin/alacli wallet keys list
```

It will display all the public key which is present in the wallet. Developer can find the relative public key.
If Developer has a wallet with specific name then the command would be a little bit different.

```
./bin/alacli wallet import --name=YOUR_WALLET_NAME
```

DApp developer needs to create one additional set of keys for set the custom permission.

→ **For creating a pair of Public and Private key**

./bin/alacli create key --to-console

When this command executes it will give you the Public and Private key. The prefix of Public key would be **ALA.** Once Developer has generated the pair of keys then he has to import their private key into their wallet. For importing the key into the wallet the steps would be the same as described above.

DApp developer has to follow the following 3 commands to set the permissions.

→ **For set permission to the account**

./bin/alacli -u **API_ENDPOINT_OF_BP** set account permission **YOUR_ACCOUNT_NAME** custom '{"threshold":1,"keys":[{"key":"**YOUR_NEW_PUB_KEY**","weight":1}],"accounts":[{"permission":{"actor":"dappregistry","permission":"alaio.code"},"weight":1}]}' -p **YOUR_ACCOUNT_NAME**

→ **For set action permission to the account**

./bin/alacli -u **API_ENDPOINT_OF_BP** set action permission **YOUR_ACCOUNT_NAME** dappregistry ontransfer custom -p **YOUR_ACCOUNT_NAME**

→ **For set account permission to the dappregistry account.**

./bin/alacli -u **API_ENDPOINT_OF_BP** set account permission dappregistry active --add-code -p dappregistry

Once, all these steps done developer must have their application smart contract and they have to set their smart contract on the network.

→ **For set developer's smart contract**

./bin/alacli -u **API_ENDPOINT_OF_BP** set contract **YOUR_ACCOUNT_NAME YOUR_CONTRACT_PATH CONTRACT_NAME.wasm** --abi **CONTRACT_NAME.abi**

→ **For set the DApp on dappregistry**

./bin/alacli -u **API_ENDPOINT_OF_BP** push action **YOUR_ACCOUNT_NAME** setdappsacc '["dappregistry"]' -p **YOUR_ACCOUNT_NAME**@active

→ **For check whether the registry is set correctly or not**

./bin/alacli -u **API_ENDPOINT_OF_BP** get table **YOUR_ACCOUNT_NAME** **YOUR_ACCOUNT_NAME** config

→ **For add apps to DApp Registry**

./bin/alacli -u **API_ENDPOINT_OF_BP** push action dappregistry add '["**YOUR_ACCOUNT_NAME**", "**YOUR_DAPP_NAME**", **YOUR_DAPP_PREFERENCE**]' -p dappregistry@active

→ **Developer can check that app is added to registry or not**

./bin/alacli -u **API_ENDPOINT_OF_BP** get table dappregistry dappregistry dapps --lower YOUR_DAPP_NAME

→ **Developer can check that DApp owner account is linked to his app or not**

./bin/alacli -u **API_ENDPOINT_OF_BP** get table dappregistry dappregistry dappaccounts -- lower **YOUR_ACCOUNT_NAME**

→ **Developers can claim their rewards with following command**

./bin/alacli -u **API_ENDPOINT_OF_BP** push action alaio claimdapprwd '["dappregistry", "**YOUR_ACCOUNT_NAME**", "**YOUR_DAPP_NAME**"]' -p **YOUR_ACCOUNT_NAME**@active

5. Voter Environment Setup

Voters must be familiar with the ALADIN's Command Line Arguments and for the User/Voter has to build and install the ALAIO package in order to participate in voting, staking and claiming his vote rewards.

If users wants to participate in voting then user must have to stake some amount of staked tokens which will help user to give votes to the Block Producers. Users can give votes upto 30 Block Producers at a time. User has to convert their liquid tokens to the staked tokens so that they can eligible to vote for the Block Producers. User can stake any number of tokens.

User/Voter will receive their Public and Private key in their email when they register themselves on ALADIN Platform. User/Voter will have to execute a command to receive the private key and import that into wallet.

→ **For generating a Private key from mnemonic code**

curl -X POST "https://api.aladinnetwork.org/users/getKeys" -H "accept: multipart/form-data" -H "Content-Type: application/x-www-form-urlencoded" -d "mnemonicCode=**MNEMONIC_CODE_RECIEVED_IN_EMAIL**&password=**PASSWORD_WHEN_YOU_REGISTER**&accountNo=1"

This Command has a parameter **"accountNo"** by default it would be 1, ALADIN facilitate the user/voters to create more than one ID with the same mnemonic code so if user/voters wants to create another ID then parameter **"accountNo"** would be 2, and user/voters will get the different set of Public and Private key.

When this command executes successfully it will give you a public & private key. User needs to create a wallet for storing the private keys and they will have to create their own public private key pair and store them into a safe place.

→ **For creating a wallet**

```
./bin/alacli create wallet --to-console
```

When this command executes successfully it will give the password of the wallet, User has to store the password in a safe place so whenever their wallet is locked they can unlock their wallet with the help of this password.By delinquency “default” wallet is created but user can create the wallet with specific name.

```
./bin/alacli create wallet --name=YOUR_WALLET_NAME --to-console
```

With the “**--name**” parameter user can create the wallet with specific name.

→ **For unlocking a wallet**

```
./bin/alacli wallet unlock
```

When this command executes it will ask for the wallet password, User has to enter their password and then wallet will be unlocked.
If user has set up their wallet with the specific name then there will be a little bit difference in unlock command.

```
./bin/alacli wallet unlock --name=YOUR_WALLET_NAME
```

→ **For importing private key into the Wallet**

```
./bin/alacli wallet import
```

It will ask for the private key, user has to enter the private key in the console which they have generated from the first step and once done, User can see his public key as output.
For cross verification

```
./bin/alacli wallet keys list
```

It will display all the public key which is present in the wallet. User can find the relative public key.
If User has a wallet with specific name then the command would be a little bit different.

```
./bin/alacli wallet import --name=YOUR_WALLET_NAME
```

→ **User has to stake some tokens to give the votes to the Block Producers**

./bin/alacli -u **API_ENDPOINT_OF_BP** system delegatebw **YOUR_ACCOUNT_NAME YOUR_ACCOUNT_NAME** “**AMOUNT** ALA” “**AMOUNT** ALA”

When this command executes successfully user can get their account information and see their liquid tokens will be deducted by the amount they have provided in the above command and that amount will be considered as staked tokens so that user can give votes to the producers.

→ **For unstaking the staked tokens**

./bin/alacli -u **API_ENDPOINT_OF_BP** system undelegatebw **YOUR_ACCOUNT_NAME YOUR_ACCOUNT_NAME** “**AMOUNT** ALA” “**AMOUNT** ALA”

Unstaking will take 3 days so user has to wait for 3 days to convert their stake balance to the liquid balance.

→ **For account information**

./bin/alacli -u **API_ENDPOINT_OF_BP** get account **YOUR_ACCOUNT_NAME**

User can check the account information by executing this command. User can see the staked amount and his liquid balance.

→ **For buy more ram resource**

./bin/alacli -u **API_ENDPOINT_OF_BP** system buyram **YOUR_ACCOUNT_NAME YOUR_ACCOUNT_NAME** “**AMOUNT** ALA”

When this command executes successfully user will get some RAM resources for their account and whatever AMOUNT of ALA tokens user has passed in the command that value will be deducted from user’s liquid balance.

→ **For vote the Block Producers**

./bin/alacli -u **API_ENDPOINT_OF_BP** system voteproducer prods YOUR_ACCOUNT_NAME BP_NAME_1 BP_NAME_2…….BP_NAME_30

User can vote upto 30 Block Producers at a time. ALADIN restricts user not to give more than 1000 votes so even if user has N number of staked tokens but at a time user can vote maximum 1000.

→ **For claiming vote rewards**

./bin/alacli -u **API_ENDPOINT_OF_BP** push action alaio claimvoterwd '["**YOUR_ACCOUNT_NAME**"]' -p **YOUR_ACCOUNT_NAME**