# Tools for Hedera

At the moment we support the following operations:

* [Generate Keys in Batch](#_Generate_Keys_in);
* [Create Account(s)](#_Create_Accounts) (and generate a simple key for this account);
* [Create an Account with a key structure](#_Create_an_Account) defined in a JSON file;
* [Update Account](#_Update_Account’s_Simple) (if is used to update key, both the new key and old key are simple keys);
* [Update Accounts’ keys from Genesis](#_Update_Accounts_from) (the new keys are auto-generated simple keys);
* Update an Account’s Multi-Key (both the new key and old key can either be simple key or multi-key). The operation contains several steps:

(1) [Generate an Unsigned Update Key Transaction](#_Generate_an_Unsigned), with the old key structure and new key structure defined in a JSON file;

(2) [Generate Signature Files](#_Generate_a_Signature) (using each simple key in the old key structure and the new key structure)

(3) [Build Multi-Sig Files](#_Build_a_Multi-Sig);

(4) [Sign the Transaction with Signature Files](#_Sign_Transaction_with)

(5) [Submit Transaction](#_Submit_Transaction)

* [Transfer Hbars from one account to another](#_Transfer_Hbars_from);
* [Get Account Info](#_Get_Account_Info);
* [Show a Transaction’s Content](#_Show_Transaction:);
* Create a File
* Update a File
* Append to a File
* Read a File

To launch the tools:

* Unzip hederatools.zip
* Move to the hederatools folder.
* Make sure that launch.sh is executable (chmod +x launch.sh)
* nodesInfo.json in the JSON\_files is required
* Tools are used with the command “./launch.sh <options>”

## Common Operations:

|  |  |  |
| --- | --- | --- |
| Operation | Command Line | Sample JSON name |
| Generate Key(s) in Batch | ./launch.sh generate-key -n {num} -f {directory name} |  |
| Create Account(s) | ./launch.sh create-account -f {json file name} -a {num} | createAccount.json |
| Create an Account with a key structure defined | ./launch.sh create-account-with-key -f {json file name} | createAccountWithSingleKey.json  createAccountWithKey\_KeyListOfSimpleKey.json  createAccountWithKey\_KeyListOfThresholdKey.json  createAccountWithKey\_ThresholdKeyOfSimpleKey.json  createAccountWithKey\_NestedThresholdKey.json |
| Update Account | ./launch.sh update-account -f {json file name} | updateAccount.json |
| Update Accounts’ keys from Genesis | ./launch.sh update-keys-in-batch -f {json file name} | batchAccountKeyUpdate.json |
| Transfer Hbars | ./launch.sh transfer-hbars -f {json file name} | createTransfer.json |
| Get an Account’s info | ./launch.sh get-account-info -f {json file name} | getAccountInfo.json |
| Generate an Unsigned Transfer Transaction | ./launch.sh generate-unsigned-transfer-transaction -f {json file name} | generateUnsignedTransfer.json |
| Generate an Unsigned Update Key Transaction | ./launch.sh generate-unsigned-update-key-transaction -f {json file name} | generateUnsignedUpdateKeyTransaction\_keyList.json  generateUnsignedUpdateKeyTransaction\_thresholdKey.json |
| Sign a Transaction file with PrivateKey | ./launch.sh sign-transaction-with-key -f {json file name} | generateSignatureFile\_Genesis.json |
| Build a Multi-Sig File | ./launch.sh build-multisig -f {json file name} | buildMultiSigFile\_SigListOfSimpleSig.json  buildMultiSigFile\_SigListOfThresholdSig.json  buildMultiSigFile\_ThresholdSigOfSimpleSig.json  buildMultiSigFile\_NestedThresholdSig.json |
| Sign the Transaction with Signature Files | ./launch.sh sign-transaction-with-sigs -f {json file name} | signTransactionWithFile.json |
| Submit the Signed Transaction | ./launch.sh submit -f {transaction file name} |  |
| Show a Transaction’s Content | ./launch.sh show-transaction -f {.proto file name} |  |
| Create a file | ./launch.sh create-file -f {JSON file name} |  |
| Update a file | ./launch.sh update-file -f {JSON file name} |  |
| Append to a file | ./launch.sh append-file -f {JSON file name} |  |
| Read a file | ./launch.sh get-file-contents -f {JSON file name} |  |

## Commands:

|  |  |
| --- | --- |
| Generate Key(s) in Batch | generate-key, gk |
| Create Account(s) | create-account, ca |
| Create an Account with a key structure | create-account-with-key, cawk |
| Update an Account’s Simple Key | update-key, uk |
| Update Accounts’key from Genesis | update-keys-in-batch, ukb |
| Create Crypto Transfer | transfer-hbars, th |
| Get an Account’s info | get-account-info, gi |
| Generate an Unsigned Transfer Transaction | generate-unsigned-transfer-transaction, ca |
| Generate an Unsigned Update Key Transaction | generate-unsigned-update-key-transaction, guuk |
| Generate a Signature File | generate-signature, gs |
| Build a Multi-Sig File | build-multisig, bm |
| Sign the Transaction with Signature Files and Submit the Signed Transaction | sign-transaction-submit, sts |
| Show a Transaction’s Content | show-transaction, show |
| Create a file | create-file, cf |
| Retrieve a file’s contents | get-file-contents, gfc |
| Update a file | update-file, uf |
| Append to a file | append-file, af |

# Generate Keys in Batch:

## Purpose:

To generate N keys stored in disk using a password

## Requirements:

* Password will be asked during execution

./launch.sh generate-key -n 7 -f <path>

## Mandatory Commands:

* -f <String> Path to the folder where keys will be stored.

## Optional Commands:

* -n <int> Number of keys generated. Default is 1

## Results:

* N keys are generated and stored separately in the location specified by the user, encrypted with a password.
* Each private key is saved as “KeyStore-id.pem”, e.g., KeyStore-0.pem;
* Each public key is saved as “PubKey-id.txt”, e.g., PubKey-0.txt;

# Update Account:

## Purpose:

To update key, receiverSigRequired, sendRecordThreshold or receiveRecordThreshold of an account.

## Requirements:

* Encrypted KeyStore must have been created prior to the execution of the command.
* The StartupAccount.txt is needed in case any of the keys is the genesis key
* If the fee payer ID is specified in the JSON, the transaction fee will be charged to that account. Otherwise it is charged to the account that will be updated.
* If no Key Locations are specified in the JSON for the fee payer or the account, the system will assume that the keys are still the genesis key.
* Fees in the JSON file are in tinybars.
* Passwords will be requested during execution.

./launch.sh update-account -f <Location of JSON>

## Mandatory Commands:

* -f <String> Location of the batch update JSON file.

## Optional Commands:

* -r <Boolean> Request record if true. Default is false.
* -l <String> Location for records. Only needed if “-r true” is set.

## Result:

* Account Key is updated to the new key.

## Update JSON file sample:

{  
 "accountIDToUpdate": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "1015"  
 },

"feePayerAccountID": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "1015"  
 },  
 "nodeAccountID": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "3"  
 },  
 "transactionValidDuration": {  
 "seconds": 30,  
 "nanos": 0  
 },  
 "transactionFee": "100000",  
 "memo": "Update an account",  
 "genesisFileName": "StartUpAccount.txt",

"receiverSigRequired": false,

"sendRecordThreshold": "10000000000",

"receiveRecordThreshold": "10000000000",  
 "locationOfCurrentKey": "KeyStore-1015.pem",  
 "locationOfNewKey": "KeyStore-1015-new.pem",

"locationOfFeePayerKey": "KeyStore-1015.pem"  
}

# Update Accounts from Genesis Keys to individual keys:

## Purpose:

To update the keys for the accounts created during Genesis.

## Requirements:

* All initial keys in the accounts must be the Genesis Key.
* The StartupAccount.txt is needed to obtain the initial key.
* All fees will be paid from the treasury account.
* JSON file must specify the first and last accounts to be updated.
* Fees in the JSON are in tinybars.
* Passwords will be requested during execution.

./launch.sh update-keys-in-batch -f <batch update JSON file>

## Mandatory Commands:

* -f <String> Location of the batch update JSON file

## Optional Commands:

* -r <Boolean> Request record if true. Default is false
* -l <String> Location for records. Only needed if “-r true” is set.

## Result:

* The folder specified in the JSON file is populated with the new keys generated for the accounts, named by appending the account ID to “KeyStore” (ex. KeyStore-51.pem)
* The keys are stored with a password which is provided by user

## Batch Update JSON file sample:

{  
 "accountIDsToUpdate": {  
 "firstAccountID": "65",  
 "lastAccountID" : "70"  
 },  
 "feePayerAccountID": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "2"  
 },  
 "nodeAccountID": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "3"  
 },  
 "transactionFee": "100000",  
 "generateRecord": "true",  
 "genesisFileName": "StartUpAccount.txt",  
 "storageLocationOfNewKeys": "./"  
}

# 

# Transfer Hbars from one account to another:

## Purpose:

To transfer tinybars from one account to another.

## Requirements:

* Encrypted KeyStore must have been created prior to the execution of the command.
* The StartupAccount.txt is needed in case any of the keys is the genesis key
* Fees in the JSON file are in tinybars.
* Passwords will be requested during execution.

./launch.sh transfer-hbars -f <Location of JSON>

## Mandatory Commands:

* -f <String> Location of the batch update JSON file.

## Optional Commands:

* -r <Boolean> Request record if true. Default is false.
* -l <String> Location for records. Only needed if “-r true” is set.

## Transfer JSON file sample:

{

"senderAccountID": {

"shardNum": "0",

"realmNum": "0",

"accountNum": "1002"

},

"feePayerAccountID": {

"shardNum": "0",

"realmNum": "0",

"accountNum": "1001"

},

"nodeAccountID": {

   "shardNum": "0",

   "realmNum": "0",

   "accountNum": "3"

},

"recipientAccountID": {

   "shardNum": "0",

   "realmNum": "0",

   "accountNum": "1003"

},

"amount": "5000",

"transactionValidDuration": {

   "seconds": 30,

   "nanos": 0

},

"transactionFee": "100000",

"generateRecord": true,

"memo": "Create a transfer",

"genesisFileName": "StartUpAccount.txt",

"locationOfFeePayerKey": "Keys/KeyStore-1001.pem",

"locationOfSenderKey": "Keys/KeyStore-1002.pem"

}

# Create Accounts

## Purpose:

To create new accounts.

## Requirements:

* The StartupAccount.txt is needed in case any of the keys is the genesis key
* Fees and balances in the JSON file are in tinybars.
* Passwords will be requested during execution.

./launch.sh create-account -f <Location of JSON>

## Mandatory Commands:

* -f <String> Location of the create-accounts JSON file.

## Optional Commands:

* -r <Boolean> Request record if true. Default is false.
* -l <String> Location for records. Only needed if “-r true” is set.
* -a <int> Number of accounts to be created (default is 1)

## Create Accounts JSON file sample:

{

"feePayerAccountID": {

"shardNum": "0",

"realmNum": "0",

"accountNum": "2"

},

"nodeAccountID": {

"shardNum": "0",

"realmNum": "0",

"accountNum": "3"

},

"transactionValidDuration": {

"seconds": 30

},

"transactionFee": "100000",

"generateRecord": true,

"memo": "Create an account",

"initialBalance": "1000000",

"proxyAccountID": {

"shardNum": "0",

"realmNum": "0",

"accountNum": "14"

},

"proxyFraction": "55",

"maxReceiveProxyFraction": "44",

"sendRecordThreshold": "111",

"receiveRecordThreshold": "333",

"receiverSigRequired": false,

"autoRenewPeriod": 15,

"shardID": "0",

"realmID":"0",

"genesisFileName": "StartUpAccount.txt",

"newKeyLocation": "./Keys"

}

# Create an Account with a key structure

## Purpose:

To create a new account with a key structure defined in a jSON file.

## Requirements:

* The StartupAccount.txt is needed in case any of the keys is the genesis key.
* Fees and balances in the JSON file are in tinybars.
* The files in the key structure are public key files. The corresponding KeyStore must have been created prior to the execution of the command.
* The value of the “key” field in the JSON file can either be a String of a public key’s file name, or a JsonObject which defines a KeyList or ThresholdKey.

Examples of a simple key, KeyList, and ThresholdKey are as following:

1. Simple key:

"key": "./PubKey-0.txt"

1. KeyList:

"key": {  
 "keyList": [  
 {  
 "key": "./PubKey-0.txt"  
 }, {  
 "key": "./PubKey-1.txt"  
 }, {

"key": "./PubKey-2.txt"  
 }  
 ]  
}

1. ThresholdKey:

"key": {  
 "thresholdKey": {  
 "threshold": 2,  
 "keyList": [  
 {  
 "key": "./PubKey-0.txt"  
 }, {  
 "key": "./PubKey-1.txt"  
 }, {  
 "key": "./PubKey-2.txt"  
 }  
 ]  
 }

}

./launch.sh create-account-with-key -f <Location of JSON>

## Mandatory Commands:

* -f <String> Location of the create-accounts JSON file.

## Optional Commands:

* -r <Boolean> Request record if true. Default is false.
* -l <String> Location for records. Only needed if “-r true” is set.

## Create an Account with a key structure JSON file sample:

{  
 "feePayerAccountID": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "2"  
 },  
 "nodeAccountID": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "3"  
 },  
 "transactionValidDuration": {  
 "seconds": 60,  
 "nanos": 0  
 },  
 "transactionFee": "100000",  
 "generateRecord": true,  
 "memo": "Create an account",  
 "key": {  
 "thresholdKey": {  
 "threshold": 2,  
 "keyList": [  
 {  
 "key": "./PubKey-0.txt"  
 }, {  
 "key": "./PubKey-1.txt"  
 }, {  
 "key": "./PubKey-2.txt"  
 }  
 ]  
 }  
 },  
 "initialBalance": "1000000",  
 "proxyAccountID": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "3"  
 },  
 "proxyFraction": "55",  
 "maxReceiveProxyFraction": "44",  
 "sendRecordThreshold": "500000000000000000",  
 "receiveRecordThreshold": "500000000000000000",  
 "receiverSigRequired": false,  
 "autoRenewPeriod": 15,  
 "shardID": "0",  
 "realmID":"0",  
 "genesisFileName": "StartUpAccount.txt"  
}

# Get Account Info

## Purpose:

To get an Account’s information.

## Requirements:

* The StartupAccount.txt is needed in case any of the keys is the genesis key
* If no Key Locations are specified in the JSON for the fee payer or the sender account, the system will assume that the keys are still the genesis key.
* Fees and balances in the JSON file are in tinybars.
* Passwords will be requested during execution.

./launch.sh get-account-info -f <Location of JSON>

## Mandatory Commands:

* -f <String> Location of the get-acccount-info JSON file.

## Get Account Info JSON file sample:

{  
 "senderAccountID": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "1013"  
 },  
 "feePayerAccountID": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "2"  
 },  
 "nodeAccountID": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "3"  
 },  
 "accountIDToGetInfo": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "99"  
 },  
 "nodeFee": "100000",  
 "transactionValidDuration": {  
 "seconds": 30,  
 "nanos": 0  
 },  
 "transactionFee": "100000",  
 "generateRecord": "true",  
 "memo": "Get Account Info",  
 "genesisFileName": "StartUpAccount.txt",  
 "locationOfPayerKey": "Keys/KeyStore-1013.pem",  
 "locationOfFeePayerKey": "genesis"  
}

# Generate an Unsigned Update Key Transaction:

## Purpose:

To generate an Unsigned Update Key Transaction, such that we can then collect signature files for this transaction from payer’s key, old key and new key, build multi-sig file if a key is a multi-key, and attach signature files to the transaction and submit the transaction.

## Requirements:

* The files in the key structure are public key files. The corresponding KeyStore must have been created prior to the execution of the command.
* The StartupAccount.txt is needed in case any of the keys is the genesis key
* Fees in the JSON file are in tinybars.

./launch.sh generate-unsigned-update-key-transaction -f <Location of JSON>

## Mandatory Commands:

* -f <String> Location of the JSON file.

## Result:

* A transaction file is saved as a .proto file.

## JSON file sample:

{  
 "accountIDToUpdate": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "1065"  
 },  
 "feePayerAccountID": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "1065"  
 },  
 "nodeAccountID": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "3"  
 },  
 "transactionValidDuration": {  
 "seconds": 120,  
 "nanos": 0  
 },  
 "transactionFee": "100000",  
 "generateRecord": "true",  
 "memo": "Update an account",  
 "genesisFileName": "StartUpAccount.txt",  
 "newKey": {  
 "thresholdKey": {  
 "threshold": 2,  
 "keyList": [  
 {  
 "key": "./PubKey-0.txt"  
 }, {  
 "key": "./PubKey-1.txt"  
 }, {  
 "key": "./PubKey-2.txt"  
 }  
 ]  
 }  
 },  
 "newTransactionLocation": "."  
}

# Generate a Signature File:

## Purpose:

To generate a Signature File for a Transaction.

## Requirements:

* The transactionFile must have been created prior to the execution of the command.
* The StartupAccount.txt is needed if the signer’s key is the genesis key
* If the signer’s key is an element of a key structure, we can specify an id for this signature, which would be included in the signature’s file name. With the id corresponding to the key’s id, we can know the position of this signature file when building a multi-sig.
* Passwords will be requested during execution.

./launch.sh sign-transaction-with-key -f <Location of JSON>

## Mandatory Commands:

* -f <String> Location of the JSON file.

## Result:

* A signature file is saved as a .sig or .sigPair file. Its file name includes the transaction ID string, the signer’s account id, and the sigID if it is specified.

## JSON file sample:

{  
 "signerAccountID": {  
 "shardNum": "0",  
 "realmNum": "0",  
 "accountNum": "1076"  
 },  
 "sigID": 0,  
 "transactionFile": "./Transaction-0.0.1076\_1547758050.tx",  
 "signerKeyFile": "./KeyStore-0.pem",

"genesisFileName": "StartUpAccount.txt",  
 "newSignatureLocation": "."  
}

# Build a Multi-Sig File:

## Purpose:

To generate a Multi-sig File for a Transaction.

If an account has a multi-key, we need to collect signatures from each simple key in this account’s key structure, then build a multi-sig from those signatures.

## Requirements:

* The signature files must have been created prior to the execution of the command.
* The signature structure should be consistent with the key structure of this account.
* The value of the “sig” field in the JSON file can either be a String of a signature’s file name, or a JsonObject which defines a SigList or ThresholdSig.

Examples of a simple sig, SigList, and ThresholdSig are as following:

1. Simple sig:

"sig": "0.0.1065\_1547765277-Signature-0.0.1065.sig"

1. SigList:

"sig": {  
 "sigList" : [  
 {  
 "sig": "0.0.1065\_1547765277-Signature-0.0.1065-0.sig"  
 }, {  
 "sig": "0.0.1065\_1547765277-Signature-0.0.1065-1.sig"  
 }, {  
 "sig": "0.0.1065\_1547765277-Signature-0.0.1065-2.sig "  
 }  
 ]  
}

1. ThresholdSig:

"sig": {  
 "thresholdSig": {  
 "sigList" : [  
 {  
 "sig": "0.0.1065\_1547765277-Signature-0.0.1065-0.sig"  
 }, {  
 "sig": "0.0.1065\_1547765277-Signature-0.0.1065-1.sig"  
 }, {  
 "sig": ""  
 }  
 ]  
 }  
 }

./launch.sh build-multisig -f <Location of JSON>

## Mandatory Commands:

* -f <String> Location of the JSON file.

## Result:

* A signature file is saved as a .sig file. Its file name is specified in the JSON file.

## JSON file sample:

{  
 "newSigLocationAndFileName": "./0.0.1065\_1547765277-Signature-0.0.1065-new.sig",  
 "sig": {  
 "thresholdSig": {  
 "sigList" : [  
 {  
 "sig": "0.0.1065\_1547765277-Signature-0.0.1065-0.sig"  
 }, {  
 "sig": "0.0.1065\_1547765277-Signature-0.0.1065-1.sig"  
 }, {  
 "sig": ""  
 }  
 ]  
 }  
 }  
}

# Sign Transaction with Signature Files:

## Purpose:

To sign an unsigned transaction with signature files and save the signed transaction.

## Requirements:

* The transactionFile must have been created prior to the execution of the command.
* The signature files must have been created prior to the execution of the command.

./launch.sh sign-transaction-with-sigs -f <Location of JSON>

## Mandatory Commands:

* -f <String> Location of the JSON file.

## Optional Commands:

* -r <Boolean> Request record if true. Default is false.
* -l <String> Location for records. Only needed if “-r true” is set.

## Result:

* A signed transaction would be saved.

## JSON file sample:

{  
 "transactionFile": "./Transaction-0.0.1065\_1547765277.tx",  
 "signatureFiles": [  
 "0.0.1065\_1547765277-Signature-0.0.1065.sig",  
 "0.0.1065\_1547765277-Signature-0.0.1065.sig",  
 "0.0.1065\_1547765277-Signature-0.0.1065-new.sig"  
 ]  
}

# Submit Transaction

## Purpose:

To submit a signed transaction.

## Requirements:

* The transactionFile must have been created and signed prior to the execution of the command.

./launch.sh submit -f <Location of Transaction File>

## Mandatory Commands:

* -f <String> Location of the Transaction file.

## Optional Commands:

* -r <Boolean> Request record if true. Default is false.
* -l <String> Location for records. Only needed if “-r true” is set.

## Result:

* A signed transaction would be submitted, the response code would be displayed.

# Show Transaction:

## Purpose:

To show content of a Transaction file.

## Requirements:

* The transactionFile must have been created prior to the execution of the command.

./launch.sh show-transaction -f <Location of transaction file>

## Mandatory Commands:

* -f <String> Location of the transaction file.

## Result:

* Content of the transaction file would be displayed.

# Create a File

## Purpose:

To create a file in the file system

## Requirements:

* The json file must have been created prior to the execution of the command.
* The contents to be stored must have been created prior to the execution of the command.

./launch.sh create-file -f <Location of JSON file>

## Mandatory Commands:

* -f <String> Location of the JSON file.

## Optional Commands:

* -r <Boolean> Request record if true. Default is false.
* -l <String> Location for records. Only needed if “-r true” is set.

## Result:

* Upon successful creation of the file, its file ID number will be displayed.

## JSON file sample:

{

"feePayerAccountID": {

  "shardNum": "0",

  "realmNum": "0",

  "accountNum": "2"

},

"nodeAccountID": {

  "shardNum": "0",

  "realmNum": "0",

  "accountNum": "3"

},

"transactionValidDuration": {

  "seconds": 30,

  "nanos": 0

},

"transactionFee": "100000",

"memo": "Create a File",

"genesisFileName": "StartUpAccount.txt",

"expirationTime": {

  "seconds": 4102466400,

  "nanos": 0

},

"contents": "testbytes.txt",

"keysLocations": [

  "keysForTesting/Dir1/KeyStore-1.pem",

  "keysForTesting/Dir1/KeyStore-2.pem",

  "keysForTesting/Dir1/KeyStore-3.pem"

]

}

# Get a File Contents

## Purpose:

To get the contents of a file in the file system

## Requirements:

* The json file must have been created prior to the execution of the command.

./launch.sh get-file-contents -f <Location of json file>

## Mandatory Commands:

* -f <String> Location of the JSON file.
* -o <String> Location where file will be stored

## Optional Commands:

* None

## Result:

* Upon success, the contents of the file will be stored in the specified location.

## Note:

* The File ID must be specified in the JSON file.

## JSON file sample:

{

"feePayerAccountID": {

  "shardNum": "0",

  "realmNum": "0",

  "accountNum": "2"

},

"nodeAccountID": {

  "shardNum": "0",

  "realmNum": "0",

  "accountNum": "3"

},

"transactionValidDuration": {

  "seconds": 30,

  "nanos": 0

},

"transactionFee": "100000",

"nodeFee": "100000",

"memo": "Get a File Contents",

"genesisFileName": "StartUpAccount.txt",

"expirationTime": {

  "seconds": 4102466400,

  "nanos": 0

},

"fileID": {

  "shardNum": "0",

  "realmNum": "0",

  "fileNum": "1029"

}

}

# Update a File

## Purpose:

To update the contents of a file, its expiration date, or its list of signatures.

## Requirements:

* The json file must have been created prior to the execution of the command.

./launch.sh update-file -f <Location of json file>

## Mandatory Commands:

* -f <String> Location of the JSON file.

## Optional Commands:

* -r <Boolean> Request record if true. Default is false.
* -l <String> Location for records. Only needed if “-r true” is set.

## Result:

* Upon success, the file will be updated.

## JSON file sample:

{

"feePayerAccountID": {

  "shardNum": "0",

  "realmNum": "0",

  "accountNum": "2"

},

"nodeAccountID": {

  "shardNum": "0",

  "realmNum": "0",

  "accountNum": "3"

},

"transactionValidDuration": {

  "seconds": 30,

  "nanos": 0

},

"transactionFee": "100000",

"memo": "Create a File",

"genesisFileName": "StartUpAccount.txt",

"fileID": {

  "shardNum": "0",

  "realmNum": "0",

  "fileNum": "1029"

},

"newExpiration": {

  "seconds": 4102466600,

  "nanos": 0

},

"keysLocations": [

  "keysForTesting/Dir1/KeyStore-1.pem",

  "keysForTesting/Dir1/KeyStore-2.pem",

  "keysForTesting/Dir1/KeyStore-3.pem"

],

  "newContents": "newtestbytes.txt",

"newKeysLocations": [

  "keysForTesting/Dir2/KeyStore-4.pem",

  "keysForTesting/Dir2/KeyStore-5.pem",

  "keysForTesting/Dir2/KeyStore-6.pem",

  "keysForTesting/Dir2/KeyStore-7.pem"

]

}

## Note:

* This command can update one or more of the following:
  + File contents,
  + File’s expiration date,
  + List of signatures

depending on the contents of the JSON file

* The File ID must be specified in the JSON file.

# Append a File Contents

## Purpose:

To get the append data to the contents of a file in the file system

## Requirements:

* The json file must have been created prior to the execution of the command.

./launch.sh append-file -f <Location of json file>

## Mandatory Commands:

* -f <String> Location of the JSON file.

## Optional Commands:

* -r <Boolean> Request record if true. Default is false.
* -l <String> Location for records. Only needed if “-r true” is set.

## Result:

* Upon success, the contents of the file will be stored in the specified location.

## Note:

* The File ID must be specified in the JSON file.

## JSON File Sample

{

"feePayerAccountID": {

  "shardNum": "0",

  "realmNum": "0",

  "accountNum": "2"

},

"nodeAccountID": {

  "shardNum": "0",

  "realmNum": "0",

  "accountNum": "3"

},

"transactionValidDuration": {

  "seconds": 30,

  "nanos": 0

},

"transactionFee": "100000",

"memo": "Append a File",

"genesisFileName": "StartUpAccount.txt",

"contents": "testbytes.txt",

"keysLocations": [

  "keysForTesting/Dir1/KeyStore-1.pem",

  "keysForTesting/Dir1/KeyStore-2.pem",

  "keysForTesting/Dir1/KeyStore-3.pem"

],

"fileID": {

"shardNum": "0",

"realmNum": "0",

"fileNum": "1029"

}

}