remme-core-cli Documentation

Release 0.2.0

Remme

Apr 04, 2019

CONTENTS

The command-line interface (CLI) that provides a set of commands to interact with Remme-core article.

Disclaimer! The big part of functionality was roughly copy pasted from original framework **Remme-core** based on to be extended with the custom commands.

References:

- Source code of the framework's command line interface — https://github.com/hyperledger/sawtooth-core/tree/master/cli
- Documentation of the framework's command line interface — https://sawtooth.hyperledger.org/docs/core/releases/latest/cli.html

CHAPTER

ONE

USER'S GUIDE

1.1 Command line interface

This chapter shows the available options and arguments for each command and subcommand. The synopsis for each command shows its parameters and their usage.

- Optional parameters are shown in square brackets
- Choices are shown in curly braces.
- User-supplied values are shown in angle brackets.

This usage information is also available on the command line by using the `-h` or `--help` option. To get the version of the command line interface, use `-V` or `--version` option.

1.1.1 remme

The `remme` command is the usual way to interact with validators or validator networks.

This command has a multi-level structure. It starts with the base call to `remme`. Next is a top-level subcommand such as `block` or `state`. Each top-level subcommand has additional subcommands that specify the operation to perform, such as `list` or `create`. The subcommands have options and arguments that control their behavior. For example:

```
$ remme state list --format csv
```

```
usage: remme [-h] [-v] [-V]
                {batch,block,identity,keygen,peer,status,settings,state,transaction}
                ...

Provides subcommands to configure, manage, and use Sawtooth components.

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         enable more verbose output
  -V, --version         display version information

subcommands:
  {batch,block,identity,keygen,peer,status,settings,state,transaction}
    batch               Displays information about batches and submit new
                        batches
    block               Displays information on blocks in the current
                        blockchain
    identity            Works with optional roles, policies, and permissions
```

(continues on next page)

(continued from previous page)

```
    keygen              Creates user signing keys
    peer                Displays information about validator peers
    status              Displays information about validator status
    settings            Displays on-chain settings
    state               Displays information on the entries in state
    transaction         Shows information on transactions in the current chain
```

remme batch

The `remme batch` subcommands display information about the Batches in the current blockchain and submit Batches to the validator via the REST API. A Batch is a group of interdependent transactions that is the atomic unit of change in remme. For more information, see "Transactions and Batches!"

```
usage: remme batch [-h] {list,show,status,submit} ...

Provides subcommands to display Batch information and submit Batches to the
validator via the REST API.

optional arguments:
  -h, --help            show this help message and exit

subcommands:
  {list,show,status,submit}
```

remme batch list

The `remme batch list` subcommand queries the specified remme REST API (default: `http://localhost:8008`) for a list of Batches in the current blockchain. It returns the id of each Batch, the public key of each signer, and the number of transactions in each Batch.

By default, this information is displayed as a white-space delimited table intended for display, but other plain-text formats (CSV, JSON, and YAML) are available and can be piped into a file for further processing.

```
usage: remme batch list [-h] [--url URL] [-u USERNAME[:PASSWORD]]
                        [-F {csv,json,yaml,default}]

Displays all information about all committed Batches for the specified validator,␣
↪including the Batch id, public keys of all signers, and number of transactions in␣
↪each Batch.

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of the validator's REST API (default:
                        http://localhost:8008)
  -u USERNAME[:PASSWORD], --user USERNAME[:PASSWORD]
                        specify the user to authorize request
  -F {csv,json,yaml,default}, --format {csv,json,yaml,default}
                        choose the output format
```

remme batch show

The `remme batch show` subcommand queries the remme REST API for a specific batch in the current blockchain. It returns complete information for this batch in either YAML (default) or JSON format. Use the `--key` option to

narrow the returned information to just the value of a single key, either from the batch or its header.

This subcommand requires the URL of the REST API (default: `http://localhost:8008`), and can specify a *username*:*password* combination when the REST API is behind a Basic Auth proxy.

```
usage: remme batch show [-h] [--url URL] [-u USERNAME[:PASSWORD]] [-k KEY]
                        [-F {yaml,json}]
                        batch_id

Displays information for the specified Batch.

positional arguments:
  batch_id              id (header_signature) of the batch

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of the validator's REST API (default:
                        http://localhost:8008)
  -u USERNAME[:PASSWORD], --user USERNAME[:PASSWORD]
                        specify the user to authorize request
  -k KEY, --key KEY     show a single property from the block or header
  -F {yaml,json}, --format {yaml,json}
                        choose the output format (default: yaml)
```

remme batch status

The `remme batch status` subcommand queries the remme REST API for the committed status of one or more batches, which are specified as a list of comma-separated Batch ids. The output is in either YAML (default) or JSON format, and includes the ids of any invalid transactions with an error message explaining why they are invalid. The `--wait` option indicates that results should not be returned until processing is complete, with an optional timeout value specified in seconds.

This subcommand requires the URL of the REST API (default: `http://localhost:8008`), and can specify a *username*:*password* combination when the REST API is behind a Basic Auth proxy.

```
usage: remme batch status [-h] [--url URL] [-u USERNAME[:PASSWORD]]
                          [--wait [WAIT]] [-F {yaml,json}]
                          batch_ids

Displays the status of the specified Batch id or ids.

positional arguments:
  batch_ids             single batch id or comma-separated list of batch ids

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of the validator's REST API (default:
                        http://localhost:8008)
  -u USERNAME[:PASSWORD], --user USERNAME[:PASSWORD]
                        specify the user to authorize request
  --wait [WAIT]         set time, in seconds, to wait for commit
  -F {yaml,json}, --format {yaml,json}
                        choose the output format (default: yaml)
```

remme batch submit

The `remme batch submit` subcommand sends one or more Batches to the remme REST API to be submitted to the validator. The input is a binary file with a binary-encoded `BatchList` protobuf, which can contain one or more batches with any number of transactions. The `--wait` option indicates that results should not be returned until processing is complete, with an optional timeout specified in seconds.

This subcommand requires the URL of the REST API (default: `http://localhost:8008`), and can specify a *username*:*password* combination when the REST API is behind a Basic Auth proxy.

```
usage: remme batch submit [-h] [--url URL] [-u USERNAME[:PASSWORD]] [-v]
                          [-V] [--wait [WAIT]] [-f FILENAME]
                          [--batch-size-limit BATCH_SIZE_LIMIT]

Sends Batches to the REST API to be submitted to the validator. The input must
be a binary file containing a binary-encoded BatchList of one or more batches
with any number of transactions.

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of the validator's REST API (default:
                        http://localhost:8008)
  -u USERNAME[:PASSWORD], --user USERNAME[:PASSWORD]
                        specify the user to authorize request
  -v, --verbose         enable more verbose output
  -V, --version         display version information
  --wait [WAIT]         set time, in seconds, to wait for batches to commit
  -f FILENAME, --filename FILENAME
                        specify location of input file
  --batch-size-limit BATCH_SIZE_LIMIT
                        set maximum batch size; batches are split for
                        processing if they exceed this size
```

remme block

The `remme block` subcommands display information about the blocks in the current blockchain.

```
usage: remme block [-h] {list,show} ...

Provides subcommands to display information about the blocks in the current
blockchain.

optional arguments:
  -h, --help   show this help message and exit

subcommands:
  {list,show}
    list       Displays information for all blocks on the current blockchain
    show       Displays information about the specified block on the current
               blockchain
```

remme block list

The `remme block list` subcommand queries the remme REST API (default: `http://localhost:8008`) for a list of blocks in the current chain. Using the `--count` option, the number of blocks returned can be configured.

It returns the id and number of each block, the public key of each signer, and the number of transactions and batches in each.

By default, this information is displayed as a white-space delimited table intended for display, but other plain-text formats (CSV, JSON, and YAML) are available and can be piped into a file for further processing.

```
usage: remme block list [-h] [--url URL] [-u USERNAME[:PASSWORD]]
                        [-F {csv,json,yaml,default}] [-n COUNT]

Displays information for all blocks on the current blockchain, including the block id␣
↪and number, public keys all of allsigners, and number of transactions and batches.

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of the validator's REST API (default:
                        http://localhost:8008)
  -u USERNAME[:PASSWORD], --user USERNAME[:PASSWORD]
                        specify the user to authorize request
  -F {csv,json,yaml,default}, --format {csv,json,yaml,default}
                        choose the output format
  -n COUNT, --count COUNT
                        the number of blocks to list
```

remme block show

The `remme block show` subcommand queries the remme REST API for a specific block in the current blockchain. It returns complete information for this block in either YAML (default) or JSON format. Using the `--key` option, it is possible to narrow the returned information to just the value of a single key, either from the block, or its header.

This subcommand requires the URL of the REST API (default: `http://localhost:8008`), and can specify a *username*:*password* combination when the REST API is behind a Basic Auth proxy.

```
usage: remme block show [-h] [--url URL] [-u USERNAME[:PASSWORD]] [-k KEY]
                        [-F {yaml,json}]
                        block_id

Displays information about the specified block on the current blockchain.

positional arguments:
  block_id              id (header_signature) of the block

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of the validator's REST API (default:
                        http://localhost:8008)
  -u USERNAME[:PASSWORD], --user USERNAME[:PASSWORD]
                        specify the user to authorize request
  -k KEY, --key KEY     show a single property from the block or header
  -F {yaml,json}, --format {yaml,json}
                        choose the output format (default: yaml)
```

remme identity

remme supports an identity system that provides an extensible role- and policy-based system for defining permissions in a way which can be used by other pieces of the architecture. This includes the existing permissioning components for transactor key and validator key; in the future, this feature may also be used by transaction family implementations.

The `remme identity` subcommands can be used to view the current roles and policy set in state, create new roles, and new policies.

Note that only the public keys stored in the setting remme.identity.allowed_keys are allowed to submit identity transactions. Use the `sawset` commands to change this setting.

```
usage: remme identity [-h] {policy,role} ...

Provides subcommands to work with roles and policies.

optional arguments:
  -h, --help     show this help message and exit

subcommands:
  {policy,role}
    policy       Provides subcommands to display existing policies and create
                 new policies
    role         Provides subcommands to display existing roles and create new
                 roles
```

remme identity policy

The `remme identity policy` subcommands are used to display the current policies stored in state and to create new policies.

```
usage: remme identity policy [-h] {create,list} ...

Provides subcommands to list the current policies stored in state and to
create new policies.

optional arguments:
  -h, --help     show this help message and exit

policy:
  {create,list}
    create       Creates batches of remme-identity transactions for setting
                 a policy
    list         Lists the current policies
```

remme identity policy create

The `remme identity policy create` subcommand creates a new policy that can then be set to a role. The policy should contain at least one "rule" (`PERMIT_KEY` or `DENY_KEY`). Note that all policies have an assumed last rule to deny all. This subcommand can also be used to change the policy that is already set to a role without having to also reset the role.

```
usage: remme identity policy create [-h] [-k KEY] [-o OUTPUT | --url URL]
                                    [--wait WAIT]
                                    name rule [rule ...]

Creates a policy that can be set to a role or changes a policy without
resetting the role.

positional arguments:
  name                  name of the new policy
```

(continues on next page)

(continued from previous page)

```
  rule                  rule with the format "PERMIT_KEY <key>" or "DENY_KEY
                        <key> (multiple "rule" arguments can be specified)

optional arguments:
  -h, --help            show this help message and exit
  -k KEY, --key KEY     specify the signing key for the resulting batches
  -o OUTPUT, --output OUTPUT
                        specify the output filename for the resulting batches
  --url URL             identify the URL of a validator's REST API
  --wait WAIT           set time, in seconds, to wait for the policy to commit
                        when submitting to the REST API.
```

remme identity policy list

The `remme identity policy list` subcommand lists the policies that are currently set in state. This list can be used to figure out which policy name should be set for a new role.

```
usage: remme identity policy list [-h] [--url URL]
                                  [--format {default,csv,json,yaml}]

Lists the policies that are currently set in state.

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of a validator's REST API
  --format {default,csv,json,yaml}
                        choose the output format
```

remme identity role

The `remme identity role` subcommands are used to list the current roles stored in state and to create new roles.

```
usage: remme identity role [-h] {create,list} ...

Provides subcommands to list the current roles stored in state and to create
new roles.

optional arguments:
  -h, --help     show this help message and exit

role:
  {create,list}
    create       Creates a new role that can be used to enforce permissions
    list         Lists the current keys and values of roles
```

remme identity role create

The `remme identity role create` subcommand creates a new role that can be used to enforce permissions. The policy argument identifies the policy that the role is restricted to. This policy must already exist and be stored in state. Use `remme identity policy list` to display the existing policies. The role name should reference an action that can be taken on the network. For example, the role named `transactor.transaction_signer` controls who is allowed to sign transactions.

```
usage: remme identity role create [-h] [-k KEY] [--wait WAIT]
                                  [-o OUTPUT | --url URL]
                                  name policy

Creates a new role that can be used to enforce permissions.

positional arguments:
  name                  name of the role
  policy                identify policy that role will be restricted to

optional arguments:
  -h, --help            show this help message and exit
  -k KEY, --key KEY     specify the signing key for the resulting batches
  --wait WAIT           set time, in seconds, to wait for a role to commit
                        when submitting to the REST API.
  -o OUTPUT, --output OUTPUT
                        specify the output filename for the resulting batches
  --url URL             the URL of a validator's REST API
```

remme identity role list

The `remme identity role list` subcommand displays the roles that are currently set in state. This list can be used to determine which permissions are being enforced on the network. The output includes which policy the roles are set to.

By default, this information is displayed as a white-space delimited table intended for display, but other plain-text formats (CSV, JSON, and YAML) are available and can be piped into a file for further processing.

```
usage: remme identity role list [-h] [--url URL]
                                [--format {default,csv,json,yaml}]

Displays the roles that are currently set in state.

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of a validator's REST API
  --format {default,csv,json,yaml}
                        choose the output format
```

remme keygen

The `remme keygen` subcommand generates a private key file and a public key file so that users can sign remme transactions and batches. These files are stored in the `<key-dir>` directory in `<key_name>.priv` and `<key_dir>/<key_name>.pub`. By default, `<key_dir>` is `~/.remme` and `<key_name>` is `$USER`.

```
usage: remme keygen [-h] [-v] [-V] [--key-dir KEY_DIR] [--force] [-q]
                    [key_name]

Generates keys with which the user can sign transactions and batches.

positional arguments:
  key_name           specify the name of the key to create

optional arguments:
```

(continues on next page)

(continued from previous page)

```
  -h, --help         show this help message and exit
  -v, --verbose      enable more verbose output
  -V, --version      display version information
  --key-dir KEY_DIR  specify the directory for the key files
  --force            overwrite files if they exist
  -q, --quiet        do not display output

The private and public key files are stored in <key-dir>/<key-name>.priv and
<key-dir>/<key-name>.pub. <key-dir> defaults to ~/.sawtooth and <key-name>
defaults to $USER.
```

remme peer

The `remme peer` subcommand displays the addresses of a specified validator's peers.

```
usage: remme peer [-h] {list} ...

Provides a subcommand to list a validator's peers

optional arguments:
  -h, --help  show this help message and exit

subcommands:
  {list}
```

remme peer list

The `remme peer list` subcommand displays the addresses of a specified validator's peers.

This subcommand requires the URL of the REST API (default: `http://localhost:8008`), and can specify a *username*:*password* combination when the REST API is behind a Basic Auth proxy.

```
usage: remme peer list [-h] [--url URL] [-u USERNAME[:PASSWORD]]
                       [-F {csv,json,yaml,default}]

Displays the addresses of the validators with which a specified validator is
peered.

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of the validator's REST API (default:
                        http://localhost:8008)
  -u USERNAME[:PASSWORD], --user USERNAME[:PASSWORD]
                        specify the user to authorize request
  -F {csv,json,yaml,default}, --format {csv,json,yaml,default}
                        choose the output format
```

remme settings

The `remme settings` subcommand displays the values of currently active on-chain settings.

```
usage: remme settings [-h] {list} ...

Displays the values of currently active on-chain settings.

optional arguments:
  -h, --help  show this help message and exit

settings:
  {list}
    list      Lists the current keys and values of on-chain settings
```

remme settings list

The `remme settings list` subcommand displays the current keys and values of on-chain settings.

```
usage: remme settings list [-h] [--url URL] [--filter FILTER]
                           [--format {default,csv,json,yaml}]

List the current keys and values of on-chain settings. The content can be
exported to various formats for external consumption.

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of a validator's REST API
  --filter FILTER       filters keys that begin with this value
  --format {default,csv,json,yaml}
                        choose the output format
```

remme state

The `remme state` subcommands display information about the entries in the current blockchain state.

```
usage: remme state [-h] {list,show} ...

Provides subcommands to display information about the state entries in the
current blockchain state.

optional arguments:
  -h, --help   show this help message and exit

subcommands:
  {list,show}
```

remme state list

The `remme state list` subcommand queries the remme REST API for a list of all state entries in the current blockchain state. This subcommand returns the address of each entry, its size in bytes, and the byte-encoded data it contains. It also returns the head block for which this data is valid.

To control the state that is returned, use the `subtree` argument to specify an address prefix as a filter or a block id to use as the chain head.

By default, this information is displayed as a white-space delimited table intended for display, but other plain-text formats (CSV, JSON, and YAML) are available and can be piped into a file for further processing.

This subcommand requires the URL of the REST API (default: `http://localhost:8008`), and can specify a *username*:*password* combination when the REST API is behind a Basic Auth proxy.

```
usage: remme state list [-h] [--url URL] [-u USERNAME[:PASSWORD]]
                        [-F {csv,json,yaml,default}] [--head HEAD]
                        [subtree]

Lists all state entries in the current blockchain.

positional arguments:
  subtree               address of a subtree to filter the list by

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of the validator's REST API (default:
                        http://localhost:8008)
  -u USERNAME[:PASSWORD], --user USERNAME[:PASSWORD]
                        specify the user to authorize request
  -F {csv,json,yaml,default}, --format {csv,json,yaml,default}
                        choose the output format
  --head HEAD           specify the id of the block to set as the chain head
```

remme state show

The `remme state show` subcommand queries the remme REST API for a specific state entry (address) in the current blockchain state. It returns the data stored at this state address and the id of the chain head for which this data is valid. This data is byte-encoded per the logic of the transaction family that created it, and must be decoded using that same logic.

This subcommand requires the URL of the REST API (default: `http://localhost:8008`), and can specify a *username*:*password* combination when the REST API is behind a Basic Auth proxy.

By default, the peers are displayed as a CSV string, but other plain-text formats (JSON, and YAML) are available and can be piped into a file for further processing.

```
usage: remme state show [-h] [--url URL] [-u USERNAME[:PASSWORD]]
                        [--head HEAD]
                        address

Displays information for the specified state address in the current blockchain.

positional arguments:
  address               address of the leaf

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of the validator's REST API (default:
                        http://localhost:8008)
  -u USERNAME[:PASSWORD], --user USERNAME[:PASSWORD]
                        specify the user to authorize request
  --head HEAD           specify the id of the block to set as the chain head
```

remme status

The `remme status` subcommands display information related to a validator's status.

```
usage: remme status [-h] {show} ...

Provides a subcommand to show a validator's status

optional arguments:
  -h, --help  show this help message and exit

subcommands:
  {show}
```

remme status show

The `remme status` subcommand displays information related to a validator's current status, including its public network endpoint and its peers.

This subcommand requires the URL of the REST API (default: `http://localhost:8008`), and can specify a *username*:*password* combination when the REST API is behind a Basic Auth proxy.

```
usage: remme status show [-h] [--url URL] [-u USERNAME[:PASSWORD]]
                         [-F {csv,json,yaml,default}]

Displays information about the status of a validator.

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of the validator's REST API (default:
                        http://localhost:8008)
  -u USERNAME[:PASSWORD], --user USERNAME[:PASSWORD]
                        specify the user to authorize request
  -F {csv,json,yaml,default}, --format {csv,json,yaml,default}
                        choose the output format
```

remme transaction

The `remme transaction` subcommands display information about the transactions in the current blockchain.

```
usage: remme transaction [-h] {list,show} ...

Provides subcommands to display information about the transactions in the
current blockchain.

optional arguments:
  -h, --help   show this help message and exit

subcommands:
  {list,show}
```

remme transaction list

The `remme transaction list` subcommand queries the remme REST API (default: `http://localhost:8008`) for a list of transactions in the current blockchain. It returns the id of each transaction, its family and version, the size of its payload, and the data in the payload itself.

By default, this information is displayed as a white-space delimited table intended for display, but other plain-text formats (CSV, JSON, and YAML) are available and can be piped into a file for further processing.

```
usage: remme transaction list [-h] [--url URL] [-u USERNAME[:PASSWORD]]
                              [-F {csv,json,yaml,default}]

Lists all transactions in the current blockchain.

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of the validator's REST API (default:
                        http://localhost:8008)
  -u USERNAME[:PASSWORD], --user USERNAME[:PASSWORD]
                        specify the user to authorize request
  -F {csv,json,yaml,default}, --format {csv,json,yaml,default}
                        choose the output format
```

remme transaction show

The `remme transaction show` subcommand queries the remme REST API for a specific transaction in the current blockchain. It returns complete information for this transaction in either YAML (default) or JSON format. Use the `--key` option to narrow the returned information to just the value of a single key, either from the transaction or its header.

This subcommand requires the URL of the REST API (default: `http://localhost:8008`), and can specify a *username*:*password* combination when the REST API is behind a Basic Auth proxy.

```
usage: remme transaction show [-h] [--url URL] [-u USERNAME[:PASSWORD]]
                              [-k KEY] [-F {yaml,json}]
                              transaction_id

Displays information for the specified transaction.

positional arguments:
  transaction_id        id (header_signature) of the transaction

optional arguments:
  -h, --help            show this help message and exit
  --url URL             identify the URL of the validator's REST API (default:
                        http://localhost:8008)
  -u USERNAME[:PASSWORD], --user USERNAME[:PASSWORD]
                        specify the user to authorize request
  -k KEY, --key KEY     show a single property from the block or header
  -F {yaml,json}, --format {yaml,json}
                        choose the output format (default: yaml)
```

1.1.2 remmeadm

The `remmeadm` command is used for Sawtooth administration tasks. The `remmeadm` subcommands create validator keys during initial configuration and help create the genesis block when initializing a validator.

```
usage: remmeadm [-h] [-v] [-V] {genesis,keygen} ...

Provides subcommands to create validator keys and create the genesis block
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help        show this help message and exit
  -v, --verbose     enable more verbose output
  -V, --version     display version information

subcommands:
  {genesis,keygen}
    genesis         Creates the genesis.batch file for initializing the
                    validator
    keygen          Generates keys for the validator to use when signing
                    blocks
```

remmeadm genesis

The `remmeadm genesis` subcommand produces a file for use during the initialization of a validator. A network requires an initial block (known as the *genesis block*) whose signature will determine the blockchain ID. This initial block is produced from a list of batches, which will be applied at genesis time.

The optional argument *input_file* specifies one or more files containing serialized `BatchList` protobuf messages to add to the genesis data. (Use a space to separate multiple files.) If no input file is specified, this command produces an empty genesis block.

The output is a file containing a serialized `GenesisData` protobuf message. This file, when placed at *sawtooth_data*`/genesis.batch`, will trigger the genesis process.

Note: The location of *sawtooth_data* depends on whether the environment variable `SAWTOOTH_HOME` is set. If it is, then *sawtooth_data* is located at `SAWTOOTH_HOME/data`. If it is not, then *sawtooth_data* is located at `/var/lib/sawtooth`.

When `remmeadm genesis` runs, it displays the path and filename of the target file where the serialized `GenesisData` is written. (Default: *sawtooth_data*`/genesis.batch`.) For example:

```
$ remmeadm genesis config.batch mktplace.batch
Generating /var/lib/sawtooth/genesis.batch
```

Use `--output` *filename* to specify a different name for the target file.

```
usage: remmeadm genesis [-h] [-v] [-V] [-o OUTPUT] [input_file [input_file ...]]

Generates the genesis.batch file for initializing the validator.

positional arguments:
  input_file            file or files containing batches to add to the
                        resulting GenesisData

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         enable more verbose output
  -V, --version         display version information
  -o OUTPUT, --output OUTPUT
                        choose the output file for GenesisData

This command generates a serialized GenesisData protobuf message and stores it
in the genesis.batch file. One or more input files (optional) can contain
```

(continues on next page)

(continued from previous page)

```
serialized BatchList protobuf messages to add to the GenesisData. The output
shows the location of this file. By default, the genesis.batch file is stored
in /var/lib/sawtooth. If $SAWTOOTH_HOME is set, the location is
$SAWTOOTH_HOME/data/genesis.batch. Use the --output option to change the name
of the file.
```

remmeadm keygen

The `remmeadm keygen` subcommand generates keys that the validator uses to sign blocks. This system-wide key must be created during Sawtooth configuration.

Validator keys are stored in the directory `/etc/sawtooth/keys/`. By default, the public-private key files are named `validator.priv` and validator.pub. Use the <key-name> argument to specify a different file name.

```
usage: remmeadm keygen [-h] [-v] [-V] [--force] [-q] [key_name]

Generates keys for the validator to use when signing blocks.

positional arguments:
  key_name       name of the key to create

optional arguments:
  -h, --help     show this help message and exit
  -v, --verbose  enable more verbose output
  -V, --version  display version information
  --force        overwrite files if they exist
  -q, --quiet    do not display output

The private and public key pair is stored in /etc/sawtooth/keys/<key-
name>.priv and /etc/sawtooth/keys/<key-name>.pub.
```

1.1.3 remmenet

The `remmenet` command is used to interact with an entire network of Sawtooth nodes.

```
usage: remmenet [-h] [-v] [-V] {compare-chains,list-blocks,peers} ...

Inspect status of a sawtooth network

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         enable more verbose output
  -V, --version         display version information

subcommands:
  {compare-chains,list-blocks,peers}
    compare-chains      Compare chains from different nodes.
    list-blocks         List blocks from different nodes.
    peers               Shows the peering arrangment of a network
```

remmenet compare-chains

The `remmenet compare-chains` subcommand compares chains across the specified nodes.

```
usage: remmenet compare-chains [-h] [-v] [-V] [--users USERNAME[:PASSWORD]]
                               [-l LIMIT] [--table] [--tree]
                               urls [urls ...]

Compute and display information about how the chains at different nodes differ.

positional arguments:
  urls                  The URLs of the validator's REST APIs of interest,
                        separated by commas or spaces. (no default)

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         enable more verbose output
  -V, --version         display version information
  --users USERNAME[:PASSWORD]
                        Specify the users to authorize requests, in the same
                        order as the URLs, separate by commas. Passing empty
                        strings between commas is supported.
  -l LIMIT, --limit LIMIT
                        the number of blocks to request at a time
  --table               Print out a fork table for all nodes since the common
                        ancestor.
  --tree                Print out a fork tree for all nodes since the common
                        ancestor.

By default, prints a table of summary data and a table of per-node data with
the following fields. Pass --tree for a fork graph.

COMMON ANCESTOR
    The most recent block that all chains have in common.

COMMON HEIGHT
    Let min_height := the minimum height of any chain across all nodes passed
    in. COMMON HEIGHT = min_height.

HEAD
    The block id of the most recent block on a given chain.

HEIGHT
    The block number of the most recent block on a given chain.

LAG
    Let max_height := the maximum height of any chain across all nodes passed
    in. LAG = max_height - HEIGHT for a given chain.

DIVERG
    Let common_ancestor_height := the height of the COMMON ANCESTOR.
    DIVERG = HEIGHT - common_ancestor_height
```

remmenet peers

```
usage: remmenet peers [-h] {list,graph} ...

Shows the peering arrangment of a network.

optional arguments:
```

(continues on next page)

(continued from previous page)

```
  -h, --help    show this help message and exit

subcommands:
  {list,graph}
    list        Lists peers for validators with given URLs
    graph       Generates a file to graph a network's peering arrangement
```

remmenet peers list

The `remmenet peers list` subcommand displays the peers of the specified nodes.

```
usage: remmenet peers list [-h] [-v] [-V] [--users USERNAME[:PASSWORD]]
                           [--pretty]
                           urls [urls ...]

Lists peers for validators with given URLs.

positional arguments:
  urls                  The URLs of the validator's REST APIs of interest,
                        separated by commas or spaces. (no default)

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         enable more verbose output
  -V, --version         display version information
  --users USERNAME[:PASSWORD]
                        Specify the users to authorize requests, in the same
                        order as the URLs, separate by commas. Passing empty
                        strings between commas is supported.
  --pretty, -p          Pretty-print the results
```

remmenet peers graph

The `remmenet peers graph` subcommand displays a file called `peers.dot` that describes the peering arrangement of the specified nodes.

```
usage: remmenet peers graph [-h] [-v] [-V] [--users USERNAME[:PASSWORD]]
                            [-o OUTPUT] [--force]
                            urls [urls ...]

Generates a file to graph a network's peering arrangement.

positional arguments:
  urls                  The URLs of the validator's REST APIs of interest,
                        separated by commas or spaces. (no default)

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         enable more verbose output
  -V, --version         display version information
  --users USERNAME[:PASSWORD]
                        Specify the users to authorize requests, in the same
                        order as the URLs, separate by commas. Passing empty
```

(continues on next page)

(continued from previous page)

```
                        strings between commas is supported.
  -o OUTPUT, --output OUTPUT
                        The path of the dot file to be produced (defaults to
                        peers.dot)
  --force               TODO
```

1.1.4 remmeset

The `remmeset` command is used to work with settings proposals.

Sawtooth supports storing settings on-chain. The `remmeset` subcommands can be used to view the current proposals, create proposals, vote on existing proposals, and produce setting values that will be set in the genesis block.

```
usage: remmeset [-h] [-v] [-V] {genesis,proposal} ...

Provides subcommands to change genesis block settings and to view, create, and
vote on settings proposals.

optional arguments:
  -h, --help          show this help message and exit
  -v, --verbose       enable more verbose output
  -V, --version       display version information

subcommands:
  {genesis,proposal}
    genesis           Creates a genesis batch file of settings transactions
    proposal          Views, creates, or votes on settings change proposals
```

remmeset genesis

The `remmeset genesis` subcommand creates a Batch of settings proposals that can be consumed by `sawadm genesis` and used during genesis block construction.

```
usage: remmeset genesis [-h] [-k KEY] [-o OUTPUT] [-T APPROVAL_THRESHOLD]
                        [-A AUTHORIZED_KEY]

Creates a Batch of settings proposals that can be consumed by "remmeadm genesis"
and used during genesis block construction.

optional arguments:
  -h, --help            show this help message and exit
  -k KEY, --key KEY     specify signing key for resulting batches and initial
                        authorized key
  -o OUTPUT, --output OUTPUT
                        specify the output file for the resulting batches
  -T APPROVAL_THRESHOLD, --approval-threshold APPROVAL_THRESHOLD
                        set the number of votes required to enable a setting
                        change
  -A AUTHORIZED_KEY, --authorized-key AUTHORIZED_KEY
                        specify a public key for the user authorized to submit
                        config transactions
```

remmeset proposal

The Settings transaction family supports a simple voting mechanism for applying changes to on-chain settings. The `remmeset proposal` subcommands provide tools to view, create and vote on proposed settings.

```
usage: remmeset proposal [-h] {create,list,vote} ...

Provides subcommands to view, create, or vote on proposed settings

optional arguments:
  -h, --help          show this help message and exit

subcommands:
  {create,list,vote}
    create            Creates proposals for setting changes
    list              Lists the currently proposed (not active) settings
    vote              Votes for specific setting change proposals
```

remmeset proposal create

The `remmeset proposal create` subcommand creates proposals for settings changes. The change may be applied immediately or after a series of votes, depending on the vote threshold setting.

```
usage: remmeset proposal create [-h] [-k KEY] [-o OUTPUT | --url URL]
                                setting [setting ...]

Create proposals for settings changes. The change may be applied immediately
or after a series of votes, depending on the vote threshold setting.

positional arguments:
  setting               configuration setting as key/value pair with the
                        format <key>=<value>

optional arguments:
  -h, --help            show this help message and exit
  -k KEY, --key KEY     specify a signing key for the resulting batches
  -o OUTPUT, --output OUTPUT
                        specify the output file for the resulting batches
  --url URL             identify the URL of a validator's REST API
```

remmeset proposal list

The `remmeset proposal list` subcommand displays the currently proposed settings that are not yet active. This list of proposals can be used to find proposals to vote on.

```
usage: remmeset proposal list [-h] [--url URL] [--public-key PUBLIC_KEY]
                              [--filter FILTER]
                              [--format {default,csv,json,yaml}]

Lists the currently proposed (not active) settings. Use this list of proposals
to find proposals to vote on.

optional arguments:
  -h, --help            show this help message and exit
```

(continues on next page)

(continued from previous page)

```
  --url URL             identify the URL of a validator's REST API
  --public-key PUBLIC_KEY
                        filter proposals from a particular public key
  --filter FILTER       filter keys that begin with this value
  --format {default,csv,json,yaml}
                        choose the output format
```

remmeset proposal vote

The `remmeset proposal vote` subcommand votes for a specific settings-change proposal. Use `remmeset proposal list` to find the proposal id.

```
usage: remmeset proposal vote [-h] [--url URL] [-k KEY]
                              proposal_id {accept,reject}

Votes for a specific settings change proposal. Use "remmeset proposal list" to
find the proposal id.

positional arguments:
  proposal_id        identify the proposal to vote on
  {accept,reject}    specify the value of the vote

optional arguments:
  -h, --help         show this help message and exit
  --url URL          identify the URL of a validator's REST API
  -k KEY, --key KEY  specify a signing key for the resulting transaction batch
```