My Hadoop Examples

HBase Shell Commands in Practice

June 19, 2015 Admin

HBase Shell Commands in Practice

In Our previous posts we have seen HBase Overview and HBase Installation, now it is the time to practice some Hbase Shell Commands to get familiarize with HBase. We will test a few Hbase shell commands in this post.

HBase Shell Usage

- Quote all names in HBase Shell such as table and column names.
- Commas delimit command parameters.
- Type <RETURN> after entering a command to run it.

- Dictionaries of configuration used in the creation and alteration of tables are Ruby Hashes. They look like this:

{'key1' => 'value1', 'key2' => 'value2', ...}

and are opened and closed with curley-braces.

- Key/values are delimited by the '=>' character combination.
- Usually keys are predefined constants such as NAME, VERSIONS, COMPRESSION, etc.
- Constants do not need to be quoted.
  Type 'Object.constants' to see a (messy) list of all constants in the environment.
- If you are using binary keys or values and need to enter them in the shell, use double-quote'd hexadecimal representation. For example:

```
hbase> get 't1', "keyx03x3fxcd"
hbase> get 't1', "key03023011"
hbase> put 't1', "testxefxff", 'f1:', "x01x33x40
```

The HBase shell is the (J)Ruby IRB with the below HBase-specific commands added.

HBase Shell Commands

HBase Shell Commands can be categorized into below types.

- HBase Shell General Commands
- Data Definition Commands
- Data Manipulation Commands
- Other HBase Shell Commands

General Commands

- **status** – shows the cluster status
- **table_help** – help on Table reference commands, scan, put, get, disable, drop etc.
- **version** – displays HBase version
- **whoami** – shows the current HBase user.

Welcome Donald Trump, Melania to White House: Watch Livestream
- Celebs & Protestors Continue To React To Trump & Election Shocker, Even Canada Throws Shots
- Donald Trump & Barack Obama Meeting — About That Whole Birther Thing ... (LIVE STREAM)
- Warren Beatty on his marriage to Annette Bening: 'The best thing ever'
- Idris Elba was apparently getting flirty with model Jourdan Dunn at the EMAs
- Ellen DeGeneres Offers Message of Hope After Election: Watch
- Blac Chyna — I'm Here to Deliver ... A Baby!!!
- Nicole Kidman on her 'alpha' daughter: 'We don't say bossy, we say leader'
- Prince Harry Makes First Appearance Since Confirming Relationship With Meghan Markle
- Recreational marijuana legalized in four more states: CA, ND, MA, likely ME

Facebook Page
MyHadoopExamples.com

```
hbase(main):001:0> help "status"
Command: status
Show cluster status. Can be 'summary', 'simple',
default is 'summary'. Examples:

hbase> status
hbase> status 'simple'
hbase> status 'summary'
hbase> status 'detailed'
```

DDL Commands

alter

add/modify/delete column families, as well as change table configuration

Add/Change column family

For example, to change or add the 'f1' column family in table 't1' from current value to keep a maximum of 5 cell VERSIONS, do:

```
hbase> alter 't1', NAME => 'f1', VERSIONS => 5
```

we can operate on several column families:

```
hbase> alter 't1', 'f1', {NAME => 'f2', IN_MEMOR
```

Delete column family

To delete the 'f1' column family in table 'ns1:t1', use one of:

```
hbase> alter 'ns1:t1', NAME => 'f1', METHOD => '
hbase> alter 'ns1:t1', 'delete' => 'f1'
```

Alter Table Properties

We can also change table-scope attributes like MAX_FILESIZE, READONLY, MEMSTORE_FLUSHSIZE, DEFERRED_LOG_FLUSH, etc. These can be put at the end; for example, to change the max size of a region to 128MB, do:

```
hbase> alter 't1', MAX_FILESIZE => '134217728'
```

alter_async

Same as alter command but does not wait for all regions to receive the schema changes.

alter_status

Gets the status of the alter command. Indicates the number of regions of the table that have received the updated schema. Examples are

```
hbase> alter_status 't1'
hbase> alter_status 'ns1:t1'
```

create

Used for Creating tables. Pass a table name, and a set of column family specifications (at least one), and, optionally, table configuration as arguments.

Examples:

1. Create a table with namespace=ns1 and table qualifier/name=t1

```
hbase> create 'ns1:t1', {NAME => 'f1', VERSIONS =
```

2. Create a table with namespace=default and table qualifier=t1

```
hbase> create 't1', {NAME => 'f1'}, {NAME => 'f2
hbase> # The above in shorthand would be the fol
hbase> create 't1', 'f1', 'f2', 'f3'
hbase> create 't1', {NAME => 'f1', VERSIONS => 1
hbase> create 't1', {NAME => 'f1', CONFIGURATION
```

3. Table configuration options can be put at the end.

```
hbase> create 'ns1:t1', 'f1', SPLITS => ['10', '
hbase> create 't1', 'f1', SPLITS => ['10', '20',
hbase> create 't1', 'f1', SPLITS_FILE => 'splits
hbase> create 't1', {NAME => 'f1', VERSIONS => 5
hbase> # Optionally pre-split the table into NUM
hbase> # SPLITALGO ("HexStringSplit", "UniformSp
hbase> create 't1', 'f1', {NUMREGIONS => 15, SPL
hbase> create 't1', 'f1', {NUMREGIONS => 15, SPL
```

4. We can also keep around a reference to the created table.

```
hbase> t1 = create 't1', 'f1'
```

Which gives a reference to the table named 't1', on which we can then call methods **t1.scan, t1.get**.

describe

Prints the schema of a table. We can also use abbreviated **'desc'** for the same thing.

```
hbase(main):005:0> desc 'blog'
Table blog is ENABLED
COLUMN FAMILIES DESCRIPTION
{NAME => 'content', DATA_BLOCK_ENCODING => 'NONE
ERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_C
```


- July 2015
- June 2015
- May 2015
- March 2015
- April 2014
- March 2014
- February 2014

Categories

- Coldfusion
- Css
- Flume
- Hadoop
- Hadoop
- Hbase
- HIVE
- Hive
- Html
- Interview Questions
- Java
- Jquery
- Map Reduce
- oozie
- PIG
- Pig
- Sqoop
- Sqoop
- Uncategorized
- WebDevelopment

```
{NAME => 'info', DATA_BLOCK_ENCODING => 'NONE', I
IONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS
2 row(s) in 0.1070 seconds

hbase(main):006:0>
```

disable

disable an existing HBase table.Disabled tables will not be deleted from HBase but they are not available for regular access. This table is excluded from the **list** command and we can not run any other command except either **enable** or **drop** commands on disabled tables. Disabling is similar to deleting the tables temporarily.

```
hbase> disable 't1'
  hbase> disable 'ns1:t1'
```

disable_all

Disable all of tables matching the given regex:

```
hbase> disable_all 't.*'
hbase> disable_all 'ns:t.*'
hbase> disable_all 'ns:.*'
```

drop

Dropping of HBase tables means deleting the tables permanently. To drop a table it must first be disabled.

```
hbase> drop 't1'
  hbase> drop 'ns1:t1'
```

drop_all

Drop all of the tables matching the given regex:

```
hbase> drop_all 't.*'
hbase> drop_all 'ns:t.*'
hbase> drop_all 'ns:.*'
```

enable

Used to enable a table which might be currently disabled.

```
hbase> enable 't1'
  hbase> enable 'ns1:t1'
```

enable_all

Enable all of the tables matching the given regex:

```
hbase> enable_all 't.*'
hbase> enable_all 'ns:t.*'
hbase> enable_all 'ns:.*'
```

exists

To check the existence of an HBase Table

```
hbase> exists 't1'
  hbase> exists 'ns1:t1'
```

get_table

Gets the given table name and return it as an actual object to be manipulated by the user.

```
hbase(main):014:0> t1 = get_table 'blog'
0 row(s) in 0.0130 seconds

=> Hbase::Table - blog
hbase(main):015:0>
```

On reference to table we can perform all actions of a table like shown below.

We can call 'put' on the table: it puts a row 'r' with column family 'cf', column 'q' and value 'v' into table t1.

```
hbase> t1.put 'r', 'cf:q', 'v'
```

To read the data out, we can scan the table with below command which will read all the rows in table 't'.

```
hbase> t1.scan
```

Essentially, any command that takes a table name can also be done via table reference. Other commands include things like: **get, delete, deleteall, get_all_columns, get_counter, count, incr**. These functions, along with
the standard JRuby object methods are also available via tab completion.

We can also do general admin actions directly on a table; things like enable, disable, flush and drop just by typing:

```
hbase> t.enable
hbase> t.flush
hbase> t.disable
hbase> t.drop
```

is_disabled

To know whether an HBase table is disabled or not.

```
hbase> is_disabled 't1'
  hbase> is_disabled 'ns1:t1'
```

is_enabled

To know whether an HBase table is enabled or not.

```
hbase> is_enabled 't1'
  hbase> is_enabled 'ns1:t1'
```

list

List all tables in hbase. Optional regular expression parameter could be used to filter the output. Examples:

```
hbase> list
hbase> list 'abc.*'
hbase> list 'ns:abc.*'
hbase> list 'ns:.*'
```

show_filters

Show all the filters in hbase. Example:

```
hbase> show_filters

ColumnPrefixFilter
TimestampsFilter
PageFilter
.....
KeyOnlyFilter
```

Namespace Commands

All below commands are self explanatory.

- alter_namespace

Alter namespace properties.

To add/modify a property:

```
hbase> alter_namespace 'ns1', {METHOD => 'set',
```

To delete a property:

```
hbase> alter_namespace 'ns1', {METHOD => 'unset'
```

- create_namespace

Create namespace; pass namespace name, and optionally a dictionary of namespace configuration.
Examples:

```
hbase> create_namespace 'ns1'
hbase> create_namespace 'ns1', {'PROERTY_NAME'=>
```

- describe_namespace

```
hbase> describe_namespace 'ns1'
```

- drop_namespace
- list_namespace
- list_namespace_tables

DML Commands

append

Appends a cell 'value' at specified table/row/column coordinates.

```
hbase> append 't1', 'r1', 'c1', 'value', ATTRIBU
hbase> append 't1', 'r1', 'c1', 'value', {VISIBI
```

The same commands also can be run on a table reference. Suppose you had a reference
t to table 't1', the corresponding command would be:

```
hbase> t.append 'r1', 'c1', 'value', ATTRIBUTES=
hbase> t.append 'r1', 'c1', 'value', {VISIBILITY
```

count

Count the number of rows in a table. Return value is the number of rows. This operation may take a LONG time (Run **'$HADOOP_HOME/bin/hadoop jar hbase.jar rowcount'** to run a counting mapreduce job). Current count is shown every 1000 rows by default. Count interval may be optionally specified. Scan caching is enabled on count scans by default. Default cache size is 10 rows. If our rows are small in size, you may want to increase this parameter. Examples:

```
hbase> count 'ns1:t1'
hbase> count 't1'
hbase> count 't1', INTERVAL => 100000
hbase> count 't1', CACHE => 1000
hbase> count 't1', INTERVAL => 10, CACHE => 1000
```

The same commands also can be run on a table reference. Suppose you had a reference t to table 't1', the corresponding commands would be:

```
hbase> t.count
hbase> t.count INTERVAL => 100000
hbase> t.count CACHE => 1000
hbase> t.count INTERVAL => 10, CACHE => 1000
```

delete

Put a delete cell value at specified table/row/column and optionally timestamp coordinates. Deletes must match the deleted cell's coordinates exactly. When scanning, a delete cell suppresses older versions. To delete a cell from 't1' at row 'r1' under column 'c1' marked with the time 'ts1', do:

```
hbase> delete 'ns1:t1', 'r1', 'c1', ts1
hbase> delete 't1', 'r1', 'c1', ts1
hbase> delete 't1', 'r1', 'c1', ts1, {VISIBILITY=
```

deleteall

Delete all cells in a given row; pass a table name, row, and optionally a column and timestamp. Examples:

```
hbase> deleteall 'ns1:t1', 'r1'
hbase> deleteall 't1', 'r1'
hbase> deleteall 't1', 'r1', 'c1'
hbase> deleteall 't1', 'r1', 'c1', ts1
hbase> deleteall 't1', 'r1', 'c1', ts1, {VISIBIL
```

get

Get row or cell contents; pass table name, row, and optionally a dictionary of column(s), timestamp, timerange and versions. Examples:

```
hbase> get 'ns1:t1', 'r1'
hbase> get 't1', 'r1'
hbase> get 't1', 'r1', {TIMERANGE => [ts1, ts2]}
hbase> get 't1', 'r1', {COLUMN => 'c1'}
hbase> get 't1', 'r1', {COLUMN => ['c1', 'c2', '
hbase> get 't1', 'r1', {COLUMN => 'c1', TIMESTAM
hbase> get 't1', 'r1', {COLUMN => 'c1', TIMERANG
hbase> get 't1', 'r1', {COLUMN => 'c1', TIMESTAM
hbase> get 't1', 'r1', {FILTER => "ValueFilter(=
hbase> get 't1', 'r1', 'c1'
hbase> get 't1', 'r1', 'c1', 'c2'
hbase> get 't1', 'r1', ['c1', 'c2']
hbsase> get 't1','r1', {COLUMN => 'c1', ATTRIBUT
hbsase> get 't1','r1', {COLUMN => 'c1', AUTHORIZ
```

Besides the default 'toStringBinary' format, 'get' also supports custom formatting by column. A user can define a FORMATTER by adding it to the column name in the get specification. The FORMATTER can be stipulated:

1. either as a org.apache.hadoop.hbase.util.Bytes method name (e.g, toInt, toString)

2. or as a custom class followed by method name: e.g. ‘c(MyFormatterClass).format’.

Example formatting cf:qualifier1 and cf:qualifier2 both as Integers:

```
hbase> get 't1', 'r1' {COLUMN => ['cf:qualifier1
'cf:qualifier2:c(org.apache.hadoop.hbase.util.By
```

Note that you can specify a FORMATTER by column only (cf:qualifer). You cannot specify a FORMATTER for all columns of a column family.

get_counter

Return a counter cell value at specified table/row/column coordinates.

```
hbase> get_counter 'ns1:t1', 'r1', 'c1'
  hbase> get_counter 't1', 'r1', 'c1'
```

incr

Increments a cell ‘value’ at specified table/row/column coordinates. To increment a cell value in table ‘ns1:t1′ or ‘t1′ at row ‘r1′ under column ‘c1′ by 1 (can be omitted) or 10 do:

```
hbase> incr 'ns1:t1', 'r1', 'c1'
hbase> incr 't1', 'r1', 'c1'
hbase> incr 't1', 'r1', 'c1', 1
hbase> incr 't1', 'r1', 'c1', 10
hbase> incr 't1', 'r1', 'c1', 10, {ATTRIBUTES=>{
hbase> incr 't1', 'r1', 'c1', {ATTRIBUTES=>{'myk
hbase> incr 't1', 'r1', 'c1', 10, {VISIBILITY=>'
```

put

Put a cell ‘value’ at specified table/row/column and optionally timestamp coordinates. To put a cell value into table ‘ns1:t1′ or ‘t1′ at row ‘r1′ under column ‘c1′ marked with the time ‘ts1′, do:

```
hbase> put 'ns1:t1', 'r1', 'c1', 'value'
hbase> put 't1', 'r1', 'c1', 'value'
hbase> put 't1', 'r1', 'c1', 'value', ts1
hbase> put 't1', 'r1', 'c1', 'value', {ATTRIBUTE
hbase> put 't1', 'r1', 'c1', 'value', ts1, {ATTR
hbase> put 't1', 'r1', 'c1', 'value', ts1, {VISI
```

scan

Scan a table; pass table name and optionally a dictionary of scanner specifications. Scanner specifications may include

one or more of: TIMERANGE, FILTER, LIMIT, STARTROW, STOPROW, TIMESTAMP, MAXLENGTH,
or COLUMNS, CACHE

If no columns are specified, all columns will be scanned. To scan all members of a column family, leave the qualifier empty as in 'col_family:'.

The filter can be specified in two ways:
1. Using a filterString
2. Using the entire package name of the filter.

Some examples:

```
hbase> scan 'hbase:meta'
hbase> scan 'hbase:meta', {COLUMNS => 'info:regi
hbase> scan 'ns1:t1', {COLUMNS => ['c1', 'c2'], 
hbase> scan 't1', {COLUMNS => ['c1', 'c2'], LIMI
hbase> scan 't1', {COLUMNS => 'c1', TIMERANGE =>
hbase> scan 't1', {REVERSED => true}
hbase> scan 't1', {FILTER => "(PrefixFilter ('ro
(QualifierFilter (>=, 'binary:xyz'))) AND (Times
hbase> scan 't1', {FILTER => org.apache.hadoop.h
```

For setting the Operation Attributes

```
hbase> scan 't1', { COLUMNS => ['c1', 'c2'], ATT
hbase> scan 't1', { COLUMNS => ['c1', 'c2'], AUT
```

For experts, there is an additional option — CACHE_BLOCKS — which switches block caching for the scanner on (true) or off (false). By default it is enabled. Examples:

```
hbase> scan 't1', {COLUMNS => ['c1', 'c2'], CACH
```

truncate

Disables, drops and recreates the specified table. After truncate of an HBase table, schema will be present but not the records.

truncate_preserve

Disables, drops and recreates the specified table while still maintaing the previous region boundaries.

Admin Commands

- assign
- balance_switch

- balancer
- catalogjanitor_enabled
- catalogjanitor_run
- catalogjanitor_switch
- close_region
- compact
- flush
- hlog_roll
- major_compact
- merge_region
- move
- split
- trace
- unassign
- zk_dump

Replication Commands

- add_peer
- disable_peer
- enable_peer
- list_peers
- list_replicated_tables
- remove_peer
- set_peer_tableCFs
- show_peer_tableCFs

Snapshot Commands

- clone_snapshot
- delete_snapshot
- list_snapshots
- rename_snapshot
- restore_snapshot
- snapshot

Security Commands

grant

Grant users specific rights.

```
grant <user> <permissions> [<@namespace> [<table>
```

permissions is either zero or more letters from the set “RWXCA”.
READ(‘R’), WRITE(‘W’), EXEC(‘X’), CREATE(‘C’), ADMIN(‘A’)

Note: A namespace must always precede with ‘@’ character.

```
hbase> grant 'bobsmith', 'RWXCA'
hbase> grant 'bobsmith', 'RWXCA', '@ns1'
hbase> grant 'bobsmith', 'RW', 't1', 'f1', 'col1
hbase> grant 'bobsmith', 'RW', 'ns1:t1', 'f1', '
```

revoke

Revoke a user’s access rights.

```
revoke <user> [<table> [<column family> [<column
```

```
hbase> revoke 'bobsmith'
hbase> revoke 'bobsmith', 't1', 'f1', 'col1'
hbase> revoke 'bobsmith', 'ns1:t1', 'f1', 'col1'
```

user_permission

Show all permissions for the particular user.

```
Syntax : user_permission <table>
hbase> user_permission
hbase> user_permission 'table1'
hbase> user_permission 'namespace1:table1'
hbase> user_permission '.*'
hbase> user_permission '^[A-C].*'
```

Visibility labels Commands

- add_labels
- clear_auths
- get_auths
- set_auths
- set_visibility

whoami

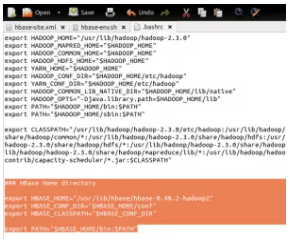
Show the current hbase user.

```
hbase> whoami
```

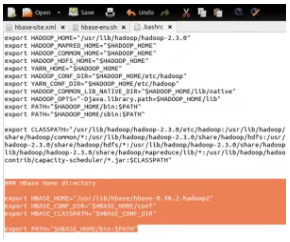
Source: http://hadooptutorial.info

Related Posts

HBase Integration with Hadoop Hive



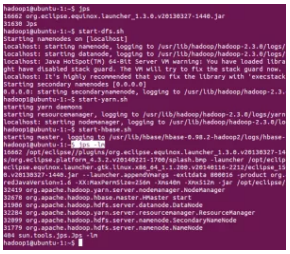
HBase Installation in Pseudo Distribution Mode



HBase Installation in Pseudo Distribution Mode

Hbase Daemons in Pseudo Distribution Mode

HBASE Interview Questions And Answers



Data Collection from HTTP Client into HBase

Share this:

More

Related Posts:

1. HBase Integration with Hadoop Hive
2. Hbase Daemons in Pseudo Distribution Mode
3. HBASE Interview Questions And Answers
4. HBase Installation in Pseudo Distribution Mode

Hbase hbase

One comment

sivanagamahesh
October 27, 2016 at 7:32 am

hi,
shell to execute various commands and perform several operations. Using these commands, we can perform multiple operations on data-tables that can give better data storage efficiencies and flexible interaction by the client. This shell commands allows the programmer to define table schemas and data operations using complete shell mode interaction . thanks for sharing me nice information.