

## HBase shell commands

As told in HBase introduction, HBase provides Extensible jruby-based (JIRB) shell as a feature to execute some commands(each command represents one functionality).

HBase shell commands are mainly categorized into 6 parts

### 1) General HBase shell commands

#### **status**

Show cluster status. Can be 'summary', 'simple', or 'detailed'. The default is 'summary'.

```
hbase> status
```

```
hbase> status 'simple'
```

```
hbase> status 'summary'
```

```
hbase> status 'detailed'
```

#### **version**

Output this HBase versionUsage:

```
hbase> version
```

#### **whoami**

Show the current hbase user.Usage:

```
hbase> whoami
```

### 2) Tables Management commands

#### **alter**

Alter column family schema; pass table name and a dictionary specifying new column family schema. Dictionaries are described on the main help command output. Dictionary must include name of column family to alter.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
```

You can operate on several column families:

```
hbase> alter 't1', 'f1', {NAME => 'f2', IN_MEMORY => true}, {NAME => 'f3', VERSIONS => 5}
```

To delete the 'f1' column family in table 't1', use one of:  
hbase> alter 't1', NAME => 'f1',  
METHOD => 'delete'

hbase> alter 't1', 'delete' => 'f1'

You can also change table-scope attributes like MAX\_FILESIZE, READONLY, MEMSTORE\_FLUSH\_SIZE, 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'

You can add a table coprocessor by setting a table coprocessor attribute:

hbase> alter 't1',  
'coprocessor'=>'hdfs:///foo.jar|com.foo.FooRegionObserver|1001|arg1=1,arg2=2'

Since you can have multiple coprocessors configured for a table, a sequence number will be automatically appended to the attribute name to uniquely identify it.

The coprocessor attribute must match the pattern below in order for the framework to understand how to load the coprocessor classes:

[coprocessor jar file location] | class name | [priority] | [arguments]

You can also set configuration settings specific to this table or column family:

hbase> alter 't1', CONFIGURATION =>  
{ 'hbase.hregion.scan.loadColumnFamiliesOnDemand' => 'true' }  
hbase> alter 't1', { NAME => 'f2', CONFIGURATION => { 'hbase.hstore.blockingStoreFiles' => '10' } }

You can also remove a table-scope attribute:

hbase> alter 't1', METHOD => 'table\_att\_unset', NAME => 'MAX\_FILESIZE'  
hbase> alter 't1', METHOD => 'table\_att\_unset', NAME => 'coprocessor\$1'

There could be more than one alteration in one command:

hbase> alter 't1', { NAME => 'f1', VERSIONS => 3 },  
{ MAX\_FILESIZE => '134217728' }, { METHOD => 'delete', NAME => 'f2' },  
OWNER => 'johndoe', METADATA => { 'mykey' => 'myvalue' }

## create

Create table; pass table name, a dictionary of specifications per column family, and optionally a dictionary of table configuration.

hbase> create 't1', { NAME => 'f1', VERSIONS => 5 }  
hbase> create 't1', { NAME => 'f1' }, { NAME => 'f2' }, { NAME => 'f3' }

hbase> # The above in shorthand would be the following:

hbase> create 't1', 'f1', 'f2', 'f3'

hbase> create 't1', {NAME => 'f1', VERSIONS => 1, TTL => 2592000, BLOCKCACHE => true}

hbase> create 't1', {NAME => 'f1', CONFIGURATION => {'hbase.hstore.blockingStoreFiles' => '10'}}

Table configuration options can be put at the end.

## **describe**

Describe the named table.

hbase> describe 't1'

## **disable**

Start disable of named table

hbase> disable 't1'

## **disable\_all**

Disable all of tables matching the given regex

hbase> disable\_all 't.\*'

## **is\_disabled**

verifies Is named table disabled

hbase> is\_disabled 't1'

## **drop**

Drop the named table. Table must first be disabled

hbase> drop 't1'

## **drop\_all**

Drop all of the tables matching the given regex

hbase> drop\_all 't.\*'

## **enable**

Start enable of named table

hbase> enable 't1'

## **enable\_all**

Enable all of the tables matching the given regex

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

## **is\_enabled**

verifies if named table enabled

```
hbase> is_enabled 't1'
```

## **exists**

Does the named table exist

```
hbase> exists 't1'
```

## **list**

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

```
hbase> list
```

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

## **show\_filters**

Show all the filters in hbase.

```
hbase> show_filters
```

## **alter\_status**

Get the status of the alter command. Indicates the number of regions of the table that have received the updated schema Pass table name.

```
hbase> alter_status 't1'
```

## **alter\_async**

Alter column family schema, does not wait for all regions to receive the schema changes. Pass table name and a dictionary specifying new column family schema. Dictionaries are described on the main help command output.

Dictionary must include name of column family to alter.

To change or add the 'f1' column family in table 't1' from defaults

to instead keep a maximum of 5 cell VERSIONS, do: `hbase> alter_async 't1', NAME => 'f1', VERSIONS => 5` To delete the 'f1' column family in table 't1', do:

```
hbase> alter_async 't1', NAME => 'f1', METHOD => 'delete' or a shorter version: hbase> alter_async 't1', 'delete' => 'f1'
```

You can also change table-scope attributes like MAX\_FILESIZE

MEMSTORE\_FLUSH\_SIZE, READONLY, and DEFERRED\_LOG\_FLUSH.

For example, to change the max size of a family to 128MB, do:

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

There could be more than one alteration in one command:

```
hbase> alter 't1', {NAME => 'f1'}, {NAME => 'f2', METHOD => 'delete'}
```

To check if all the regions have been updated, use `alter_status`

### 3) Data Manipulation commands

#### 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 your rows are small in size, you may want to increase this

parameter. Examples: `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 't1', 'r1', 'c1', ts1
```

The same command can also be run on a table reference. Suppose you had a reference `t` to table 't1', the corresponding command would be: `hbase> t.delete 'r1', 'c1', ts1`

#### deleteall

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

```
hbase> deleteall 't1', 'r1'
```

```
hbase> deleteall 't1', 'r1', 'c1', ts1
```

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.deleteall 'r1', 'c1'
```

```
hbase> t.deleteall 'r1', 'c1', ts1
```

## get

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

```
hbase> get 't1', 'r1'
```

```
hbase> get 't1', 'r1', {TIMERANGE => [ts1, ts2]}
```

```
hbase> get 't1', 'r1', {COLUMN => 'c1'}
```

```
hbase> get 't1', 'r1', {COLUMN => ['c1', 'c2', 'c3']}
```

```
hbase> get 't1', 'r1', {COLUMN => 'c1', TIMESTAMP => ts1}
```

```
hbase> get 't1', 'r1', {COLUMN => 'c1', TIMERANGE => [ts1, ts2], VERSIONS => 4}
```

```
hbase> get 't1', 'r1', {COLUMN => 'c1', TIMESTAMP => ts1, VERSIONS => 4}
```

```
hbase> get 't1', 'r1', {FILTER => "ValueFilter(=, 'binary:abc)'"}
```

```
hbase> get 't1', 'r1', 'c1'
```

```
hbase> get 't1', 'r1', 'c1', 'c2'
```

```
hbase> get 't1', 'r1', ['c1', 'c2']
```

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`)

1. 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:toInt',  
'cf:qualifier2:c(org.apache.hadoop.hbase.util.Bytes).toInt'] }
```

Note that you can specify a FORMATTER by column only (`cf:qualifier`). You cannot specify a FORMATTER for all columns of a column family. The same commands also can be run on a reference to a table (obtained via `get_table` or `create_table`). Suppose you had a reference `t` to table 't1', the corresponding commands would be:

```
hbase> t.get 'r1'
```

```
hbase> t.get 'r1', {TIMERANGE => [ts1, ts2]}
hbase> t.get 'r1', {COLUMN => 'c1'}
hbase> t.get 'r1', {COLUMN => ['c1', 'c2', 'c3']}
hbase> t.get 'r1', {COLUMN => 'c1', TIMESTAMP => ts1}
hbase> t.get 'r1', {COLUMN => 'c1', TIMERANGE => [ts1, ts2], VERSIONS => 4}
hbase> t.get 'r1', {COLUMN => 'c1', TIMESTAMP => ts1, VERSIONS => 4}
hbase> t.get 'r1', {FILTER => "ValueFilter(=, 'binary:abc')"}
hbase> t.get 'r1', 'c1'
hbase> t.get 'r1', 'c1', 'c2'
hbase> t.get 'r1', ['c1', 'c2']
```

## **get\_counter**

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

A cell should be managed with atomic increment function on HBase and the data should be binary encoded. Example:

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

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.get_counter 'r1', 'c1'
```

## **incr**

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

To increment a cell value in table 't1' at row 'r1' under column 'c1' by 1 (can be omitted) or 10 do:

```
hbase> incr 't1', 'r1', 'c1'
hbase> incr 't1', 'r1', 'c1', 1
hbase> incr 't1', 'r1', 'c1', 10
```

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.incr 'r1', 'c1'
hbase> t.incr 'r1', 'c1', 1
hbase> t.incr 'r1', 'c1', 10
```

## **put**

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

```
hbase> put 't1', 'r1', 'c1', 'value', ts1
```

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.put 'r1', 'c1', 'value', ts1
```

## 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. CACHEIf 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 – more information on this is available in the Filter Language document attached to the HBASE-4176 JIRA
2. Using the entire package name of the filter. Some examples:  

```
hbase> scan '.META.'
```

```
hbase> scan '.META.', {COLUMNS => 'info:regioninfo'}
```

```
hbase> scan 't1', {COLUMNS => ['c1', 'c2'], LIMIT => 10, STARTROW => 'xyz'}
```

```
hbase> scan 't1', {COLUMNS => 'c1', TIMERANGE => [1303668804, 1303668904]}
```

```
hbase> scan 't1', {FILTER => "(PrefixFilter ('row2') AND  
(QualifierFilter (>=, 'binary:xyz')) AND (TimestampsFilter ( 123, 456)))"}
```

```
hbase> scan 't1', {FILTER =>  
org.apache.hadoop.hbase.filter.ColumnPaginationFilter.new(1, 0)}
```

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'],  
CACHE_BLOCKS => false}
```

Also for experts, there is an advanced option — `RAW` — which instructs the scanner to return all cells (including delete markers and uncollected deleted cells). This option cannot be combined with requesting specific COLUMNS.

Disabled by default. Example:

```
hbase> scan 't1', {RAW => true, VERSIONS => 10}
```

Besides the default 'toStringBinary' format, 'scan' supports custom formatting by column. A user can define a `FORMATTER` by adding it to the column name in the scan 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> scan 't1', {COLUMNS => ['cf:qualifier1:toInt',
```



```
'cf:qualifier2:c(org.apache.hadoop.hbase.util.Bytes).toInt'] }
```

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

Scan can also be used directly from a table, by first getting a reference to a table, like such:

```
hbase> t = get_table 't'
hbase> t.scan
```

Note in the above situation, you can still provide all the filtering, columns, options, etc as described above.

## **truncate**

Disables, drops and recreates the specified table.

Examples:

```
hbase>truncate 't1'
```

## **4) HBase surgery tools**

**assign** Assign a region. Use with caution. If region already assigned, this command will do a force reassign. For experts only.

Examples:

```
hbase> assign 'REGION_NAME'
```

**balancer** Trigger the cluster balancer. Returns true if balancer ran and was able to tell the region servers to unassign all the regions to balance (the re-assignment itself is async).

Otherwise false (Will not run if regions in transition).

Examples:

```
hbase> balancer
```

**balance\_switch** Enable/Disable balancer. Returns previous balancer state.

Examples:

```
hbase> balance_switch true
```

```
hbase> balance_switch false
```

**close\_region** Close a single region. Ask the master to close a region out on the cluster or if 'SERVER\_NAME' is supplied, ask the designated hosting regionserver to close the region directly. Closing a region, the master expects 'REGIONNAME' to be a fully qualified region name. When asking the hosting regionserver to directly close a region, you pass the regions' encoded name only. A region name looks like

this:TestTable,0094429456,1289497600452.527db22f95c8a9e0116f0cc13c680396.The

trailing period is part of the regionserver name. A region's encoded name is the hash at the end of a region name; e.g. 527db22f95c8a9e0116f0cc13c680396 (without the period). A 'SERVER\_NAME' is its host, port plus startcode. For example: host187.example.com,60020,1289493121758 (find servername in master ui or when you do detailed status in shell). This command will end up running close on the region hosting regionserver. The close is done without the master's involvement (It will not know of the close). Once closed, region will stay closed. Use assign to reopen/reassign. Use unassign or move to assign the region elsewhere on cluster. Use with caution. For experts only.

Examples:hbase> close\_region 'REGIONNAME'

hbase> close\_region 'REGIONNAME', 'SERVER\_NAME'

compact Compact all regions in passed table or pass a region row to compact an individual region. You can also compact a single column family within a region.

Examples:

Compact all regions in a table:

hbase> compact 't1'

Compact an entire region:

hbase> compact 'r1'

Compact only a column family within a region:

hbase> compact 'r1', 'c1'

Compact a column family within a table:

hbase> compact 't1', 'c1'

flush Flush all regions in passed table or pass a region row to flush an individual region. For example:hbase> flush 'TABLENAME'

hbase> flush 'REGIONNAME'

major\_compact Run major compaction on passed table or pass a region row to major compact an individual region. To compact a single column family within a region specify the region name followed by the column family name.

Examples:

Compact all regions in a table:

hbase> major\_compact 't1'

Compact an entire region:

hbase> major\_compact 'r1'

Compact a single column family within a region:

hbase> major\_compact 'r1', 'c1'

Compact a single column family within a table:

hbase> major\_compact 't1', 'c1'

move Move a region. Optionally specify target regionserver else we choose one

at random. NOTE: You pass the encoded region name, not the region name so this command is a little different to the others. The encoded region name is the hash suffix on region names: e.g. if the region name were TestTable,0094429456,1289497600452.527db22f95c8a9e0116f0cc13c680396. then the encoded region name portion is 527db22f95c8a9e0116f0cc13c680396

A server name is its host, port plus startcode. For example:

host187.example.com,60020,1289493121758

Examples:hbase> move 'ENCODED\_REGIONNAME'

hbase> move 'ENCODED\_REGIONNAME', 'SERVER\_NAME'

split Split entire table or pass a region to split individual region. With the second parameter, you can specify an explicit split key for the region.

Examples:

split 'tableName'

split 'regionName' # format: 'tableName,startKey,id'

split 'tableName', 'splitKey'

split 'regionName', 'splitKey'

unassign Unassign a region. Unassign will close region in current location and then reopen it again. Pass 'true' to force the unassignment ('force' will clear all in-memory state in master before the reassign. If results in double assignment use hbck -fix to resolve. To be used by experts).

Use with caution. For expert use only. Examples:hbase> unassign 'REGIONNAME'

hbase> unassign 'REGIONNAME', true

hlog\_roll Roll the log writer. That is, start writing log messages to a new file.

The name of the regionserver should be given as the parameter. A

'server\_name' is the host, port plus startcode of a regionserver. For

example: host187.example.com,60020,1289493121758 (find servername in master ui or when you do detailed status in shell)

hbase>hlog\_roll

zk\_dump Dump status of HBase cluster as seen by ZooKeeper. Example:

hbase>zk\_dump

## 5) Cluster replication tools

add\_peer Add a peer cluster to replicate to, the id must be a short and the cluster key is composed like this:

hbase.zookeeper.quorum:hbase.zookeeper.property.clientPort:zookeeper.znode.parent

This gives a full path for HBase to connect to another cluster.

Examples:hbase> add\_peer '1', "server1.cie.com:2181:/hbase"

hbase> add\_peer '2', "zk1,zk2,zk3:2182:/hbase-prod"

remove\_peer Stops the specified replication stream and deletes all the meta

information kept about it. Examples:

```
hbase> remove_peer '1'
```

`list_peers` List all replication peer clusters.

```
hbase> list_peers
```

`enable_peer` Restarts the replication to the specified peer cluster, continuing from where it was disabled.Examples:

```
hbase> enable_peer '1'
```

`disable_peer` Stops the replication stream to the specified cluster, but still keeps track of new edits to replicate.Examples:

```
hbase> disable_peer '1'
```

`start_replication` Restarts all the replication features. The state in which each stream starts in is undetermined.

**WARNING:**

start/stop replication is only meant to be used in critical load situations.

Examples:

```
hbase> start_replication
```

`stop_replication` Stops all the replication features. The state in which each stream stops in is undetermined.

**WARNING:**

start/stop replication is only meant to be used in critical load situations.

Examples:

```
hbase> stop_replication
```

## 6) Security tools

`grant` Grant users specific rights.

Syntax : `grantpermissions` is either zero or more letters from the set "RWXCA".

`READ('R'), WRITE('W'), EXEC('X'), CREATE('C'), ADMIN('A')`For example:`hbase> grant 'bobsmith', 'RWXCA'`

```
hbase> grant 'bobsmith', 'RW', 't1', 'f1', 'col1'
```

`revoke` Revoke a user's access rights.

Syntax : `revoke`

For example:

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

`user_permission` Show all permissions for the particular user.

Syntax : `user_permission`

For example:`hbase> user_permission`

```
hbase> user_permission 'table1'
```