CRM CLI (command line interface) tool

Dejan Muhamedagic, Yan Gao dejan@suse.de,ygao@novell.com

September 23, 2009

Revision: 0.93

The CRM (a.k.a Pacemaker) is a Cluster Resource Manager which implements the cluster configuration provided by the user in CIB (Cluster Information Base). The CIB is a set of instructions coded in XML. Editing the CIB is a challenge, not only due to its complexity and a wide variety of options, but also because XML is more computer than user friendly.

> **Note**
> I do understand that there are people capable of dealing with XML without an intermediary.

There are currently three options to manage the CIB, listed here in a decreasing order of user-friendliness:

- the GUI (`hb_gui`)
- a set of command line tools
- `cibadmin(8)`

The GUI is very popular and it has recently seen a lot of good development. For some it is going to be (or remain) the first choice in cluster management.

The command line tools, lead by `crm_resource(8)`, are capable of performing almost any kind of CIB transformation. The usage is, however, plagued by the notorious weakness common to all UNIX tools: a multitude of options, necessary for operation and yet very hard to remember. Usage is also inconsistent at times.

The `cibadmin` is the ultimate CIB management tool: it applies chunks of XML written by the user or generated by another tool to the CIB. Very difficult to use without extensive training. Or should I say drill. May be unnerving as well, in particular due to sometimes cryptic error messages.

1 Design goals

The CLI provides a consistent and unified interface to CIB/cluster management. It uses the command line tools where possible and may resort to XML and `cibadmin` when there is no other option. That is the easiest way to ensure compatibility between different management tools.

It may be used either as an interactive shell or for single commands directly on the shell's command line. It is also possible to feed it a set of commands from standard input or a file, thus turning it into a scripting tool. Templates with ready made configurations may help people to learn about the cluster configuration or facilitate testing procedures.

The CLI may also be used for the CIB description and generation. A file containing a set of CLI instructions may be applied to the CLI tool to generate a complete CIB.

The new shadow CIB feature may also be put to use. The user may work on one of the shadow CIBs and then apply (or commit) it in a single step to the cluster.

It should also allow deployment of raw XML which may come either from files or network.

Several modes of operation are available to restrict the set of features depending on the user's proficiency.

The CLI is line oriented: every command must start and finish on the same line. It is possible to use a continuation character (\) to write one command in two or more lines.

The CLI has to be run on one of the cluster nodes.

> **Note**
> Even though all sensible configurations (and most of those that are not) are going to be supported by the CLI, I suspect that it may still happen that certain XML constructs may confuse the tool. When that happens, please file a bug report. The CLI will not try to update the objects it does not understand. Of course, it is always possible to edit such objects in the XML format.

2 Introduction to the user interface

Arguably the most important aspect of such a program is the user interface. We begin with an informal introduction so that the reader may get acquainted with it and get a general feeling of the tool. It is probably best just to give some examples:

1. Command line (one-shot) use:

   ```
   # crm resource stop www_app
   ```

2. Interactive use:

   ```
   # crm
   crm(live)# resource
   crm(live) resource# unmanage tetris_1
   crm(live) resource# up
   crm(live)# node standby node4
   ```

3. Cluster configuration:

   ```
   # crm<<EOF
   configure
     erase
     #
     # resources
     #
     primitive disk0 iscsi \
       params portal=192.168.2.108:3260 target=iqn.2008-07.com.suse:disk0
     primitive fs0 Filesystem \
       params device=/dev/disk/by-label/disk0 directory=/disk0 fstype=ext3
     primitive internal_ip IPaddr params ip=192.168.1.101
   ```

```
  primitive apache apache \
    params configfile=/disk0/etc/apache2/site0.conf
  primitive apcfence stonith:apcsmart \
    params ttydev=/dev/ttyS0 hostlist="node1 node2" \
    op start timeout=60s
  primitive pingd pingd \
    params name=pingd dampen=5s multiplier=100 host_list="r1 r2"
  #
  # monitor apache and the UPS
  #
  monitor apache 60s:30s
  monitor apcfence 120m:60s
  #
  # cluster layout
  #
  group internal_www \
    disk0 fs0 internal_ip apache
  clone fence apcfence \
    meta globally-unique=false clone-max=2 clone-node-max=1
  clone conn pingd \
    meta globally-unique=false clone-max=2 clone-node-max=1
  location node_pref internal_www \
    rule 50: #uname eq node1 \
    rule pingd: defined pingd
  #
  # cluster properties
  #
  property stonith-enabled=true
  commit
EOF
```

If you've ever done a CRM style configuration, you should be able to understand the above examples without much difficulties. The CLI should provide a means to manage the cluster efficiently or put together a configuration in a concise manner.

The `(live)` string in the prompt signifies that the current CIB in use is the cluster live configuration. It is also possible to work with the so-called shadow CIBs, i.e. configurations which are stored in files and aren't active, but may be applied at any time to the cluster.

Since the CIB is hierarchical such is the interface too. There are several levels and entering each of them enables the user to use a certain set of commands.

2.1 Shadow CIB usage

Shadow CIB is a new feature. Shadow CIBs may be manipulated in the same way like the *live* CIB, but these changes have no effect on the cluster resources. The administrator may choose to apply any of them to the cluster, thus replacing the running configuration with the one which is in the shadow. The `crm` prompt always contains the name of the configuration which is currently in use. Note that, for obvious reasons, only commands at the `configure` level make sense while working with a shadow CIB.

No changes take place before the `configure commit` command. Sometimes though, the administrator may start working with the running configuration, but change mind and instead of committing the changes to the cluster save them to a shadow CIB. This short `configure` session excerpt shows how:

```
crm(live)configure# cib new test-2
INFO: test-2 shadow CIB created
crm(test-2)configure# commit
```

2.2 Templates

Templates are ready made configurations created by cluster experts. They are designed in such a way, so that users may generate valid cluster configurations with minimum effort. If you are new to Pacemaker/CRM, templates may be the best way to start.

We will show here how to create a simple yet functional Apache configuration:

```
# crm configure
crm(live)configure# template
crm(live)configure template# list templates
apache       filesystem   virtual-ip
crm(live)configure template# new web <TAB><TAB>
apache       filesystem   virtual-ip
crm(live)configure template# new web apache
INFO: pulling in template apache
INFO: pulling in template virtual-ip
crm(live)configure template# list
web2-d       web2         vip2         web3         vip          web
```

We enter the `template` level from `configure`. Use the `list` command to show templates available on the system. The `new` command creates a configuration from the `apache` template. You can use tab completion to pick templates. Note that the apache template depends on a virtual IP address which is automatically pulled along. The `list` command shows the just created `web` configuration, among other configurations (I hope that you, unlike me, will use more sensible and descriptive names).

The `show` command, which displays the resulting configuration, may be used to get an idea about the minimum required changes which have to be done. All `ERROR` messages show the line numbers in which the respective parameters are to be defined:

```
crm(live)configure template# show
ERROR: 23: required parameter ip not set
ERROR: 61: required parameter id not set
ERROR: 65: required parameter configfile not set
crm(live)configure template# edit
```

The `edit` command invokes the preferred text editor with the `web` configuration. At the top of the file, the user is advised how to make changes. A good template should require from the user to specify only parameters. For example, the `web` configuration we created above has the following required and optional parameters (all parameter lines start with %%):

```
$ grep -n ^%% ~/.crmconf/web
23:%% ip
31:%% netmask
35:%% lvs_support
61:%% id
65:%% configfile
71:%% options
76:%% envfiles
```

These lines are the only ones that should be modified. Simply append the parameter value at the end of the line. For instance, after editing this template, the result could look like this (we used tabs instead of spaces to make the values stand out):

```
$ grep -n ^%% ~/.crmconf/web
23:%% ip                192.168.1.101
31:%% netmask
35:%% lvs_support
61:%% id                websvc
65:%% configfile        /etc/apache2/httpd.conf
71:%% options
76:%% envfiles
```

As you can see, the parameter line format is very simple:

```
%% <name> <value>
```

After editing the file, use `show` again to display the configuration:

```
crm(live)configure template# show
primitive virtual-ip ocf:heartbeat:IPaddr \
        params ip="192.168.1.101"
primitive apache ocf:heartbeat:apache \
        params configfile="/etc/apache2/httpd.conf"
monitor apache 120s:60s
group websvc \
        apache virtual-ip
```

The target resource of the apache template is a group which we named `websvc` in this sample session.

This configuration looks exactly as you could type it at the `configure` level. The point of templates is to save you some typing. It is important, however, to understand the configuration produced.

Finally, the configuration may be applied to the current crm configuration (note how the configuration changed slightly, though it is still equivalent, after being digested at the `configure` level):

```
crm(live)configure template# apply
crm(live)configure template# cd ..
crm(live)configure# show
node xen-b
node xen-c
primitive apache ocf:heartbeat:apache \
    params configfile="/etc/apache2/httpd.conf" \
    op monitor interval="120s" timeout="60s"
primitive virtual-ip ocf:heartbeat:IPaddr \
    params ip="192.168.1.101"
group websvc apache virtual-ip
```

Note that this still does not commit the configuration to the CIB which is used in the shell, either the running one (`live`) or some shadow CIB. For that you still need to execute the `commit` command.

We should also define the preferred node to run the service:

```
crm(live)configure# location websvc-pref websvc 100: xen-b
```

If you are not happy with some resource names which are provided by default, you can rename them now:

```
crm(live)configure# rename virtual-ip intranet-ip
crm(live)configure# show
node xen-b
```

```
node xen-c
primitive apache ocf:heartbeat:apache \
        params configfile="/etc/apache2/httpd.conf" \
        op monitor interval="120s" timeout="60s"
primitive intranet-ip ocf:heartbeat:IPaddr \
        params ip="192.168.1.101"
group websvc apache intranet-ip
location websvc-pref websvc 100: xen-b
```

To summarize, working with templates typically consists of the following steps:

- `new`: create a new configuration from templates
- `edit`: define parameters, at least the required ones
- `show`: see if the configuration is valid
- `apply`: apply the configuration to the `configure` level

2.3 Tab completion

The `crm` makes extensive use of the tab completion of `readline`. The completion is both static (i.e. for `crm` commands) and dynamic. The latter takes into account the current status of the cluster or information from installed resource agents. Sometimes, completion may also be used to get short help on resource parameters. Here a few examples:

```
crm(live)# resource <TAB><TAB>
bye         exit        manage      param       show        unmanage
cd          failcount   meta        quit        start       unmigrate
cleanup     help        migrate     refresh     status      unmove
end         list        move        reprobe     stop        up
crm(live)# configure
crm(live)configure# primitive fence-1 <TAB><TAB>
heartbeat:  lsb:        ocf:        stonith:
crm(live)configure# primitive fence-1 stonith:ipmilan params <TAB><TAB>
auth=      hostname=  ipaddr=    login=     password=  port=      priv=
crm(live)configure# primitive fence-1 stonith:ipmilan params auth=<TAB><TAB>
auth* (string)
    The authorization type of the IPMI session ("none", "straight", "md2", or "md5")
crm(live)configure# primitive fence-1 stonith:ipmilan params auth=
```

2.4 Configuration semantic checks

Resource definitions may be checked against the meta-data provided with the resource agents. These checks are currently carried out:

- are required parameters set
- existence of defined parameters
- timeout values for operations

The parameter checks are obvious and need no further explanation. Failures in these checks are treated as configuration errors.

The timeouts for operations should be at least as long as those recommended in the meta-data. Too short timeout values are a common mistake in cluster configurations and, even worse, they often slip through if cluster testing was not thorough. Though operation timeouts issues are treated as warnings, make sure that the timeouts are usable in your environment. Note also that the values given are just *advisory minimum*—your resources may require longer timeouts.

User may tune the frequency of checks and the treatment of errors by the `check-frequency` and `check-mode` preferences.

Note that if the `check-frequency` is set to `always` and the `check-mode` to `strict`, errors are not tolerated and such configuration cannot be saved.

3 Reference

We define a small and simple language. Most commands consist of just a list of simple tokens. The only complex constructs are found at the `configure` level.

The syntax is described in a somewhat informal manner: `<>` denotes a string, `[]` means that the construct is optional, the ellipsis (`...`) signifies that the previous construct may be repeated, `|` means pick one of many, and the rest are literals (strings, `:`, `=`).

3.1 `cib` (shadow CIBs)

This level is for management of shadow CIBs. It is available both at the top level and the `configure` level.

All the commands are implemented using `cib_shadow(8)` and the `CIB_shadow` environment variable. The user prompt always includes the name of the currently active shadow or the live CIB.

3.1.1 `list`

List existing shadow CIBs.

Usage:

```
list
```

3.1.2 new/delete

Create a new shadow CIB or delete an existing one. On `new`, the live cluster configuration is copied.

Usage:

```
new <cib>
delete <cib>
```

3.1.3 reset

Copy the current cluster configuration into the shadow CIB.

Usage:

```
reset <cib>
```

3.1.4 use

Choose a CIB. Leave out the CIB name to switch to the running CIB.

Usage:

```
use [<cib>]
```

3.1.5 diff

Print differences between the current cluster configuration and the active shadow CIB.

Usage:

```
diff
```

3.1.6 commit

Apply a shadow CIB to the cluster.

Usage:

```
commit <cib>
```

3.2 ra

This level contains commands which show various information about the installed resource agents. It is available both at the top level and at the `configure` level.

3.2.1 `classes`

Print all resource agents' classes and, where appropriate, a list of available providers.

Usage:

```
classes
```

3.2.2 `list`

List available resource agents for the given class. If the class is `ocf`, supply a provider to get agents which are available only from that provider.

Usage:

```
list <class> [<provider>]
```

Example:

```
list ocf pacemaker
```

3.2.3 `meta`

Show the meta-data of a resource agent type. This is where users can find information on how to use a resource agent.

Usage:

```
meta <type> <class> [<provider>]
```

Example:

```
meta apache ocf
meta ipmilan stonith
```

3.2.4 `providers`

List providers for a resource agent type.

Usage:

```
providers <type>
```

Example:

```
providers apache
```

3.3 resource

At this level resources may be managed.

All (or almost all) commands are implemented with the CRM tools such as `crm_resource(8)`.

3.3.1 status (show, list)

Print resource status. If the resource parameter is left out status of all resources is printed.

Usage:

```
status [<rsc>]
```

3.3.2 start/stop

Start/stop a resource using the `target-role` attribute.

Usage:

```
start <rsc>
stop <rsc>
```

3.3.3 manage/unmanage

Manage/unmanage a resource using the `is-managed` attribute.

Usage:

```
manage <rsc>
unmanage <rsc>
```

3.3.4 migrate/unmigrate (move/unmove)

Migrate a resource to a different node or remove the constraint generated by the previous migrate command. If node is left out, the resource is migrated by creating a constraint which prevents it from running on the current node.

Usage:

```
migrate <rsc> [<node>]
unmigrate <rsc>
```

3.3.5 `param`

Show/edit/delete a parameter of a resource.

Usage:

```
param <rsc> set <param> <value>
param <rsc> delete <param>
param <rsc> show <param>
```

Example:

```
param ip_0 show ip
```

3.3.6 `meta`

Show/edit/delete a meta attribute of a resource. Currently, all meta attributes of a resource may be managed with other commands such as `resource stop`.

Usage:

```
meta <rsc> set <attr> <value>
meta <rsc> delete <attr>
meta <rsc> show <attr>
```

Example:

```
meta ip_0 set target-role stopped
```

3.3.7 `failcount`

Show/edit/delete the failcount of a resource.

Usage:

```
failcount <rsc> set <node> <value>
failcount <rsc> delete <node>
failcount <rsc> show <node>
```

Example:

```
failcount fs_0 delete node2
```

3.3.8 cleanup

Cleanup resource status. Typically done after the resource has temporarily failed. If a node is omitted, cleanup on all nodes. If there are many nodes, the command may take a while.

Usage:

```
cleanup <rsc> [<node>]
```

3.3.9 refresh

Refresh CIB from the LRM status.

Usage:

```
refresh [<node>]
```

3.3.10 reprobe

Probe for resources not started by the CRM.

Usage:

```
reprobe [<node>]
```

3.4 node

Node management and status commands.

3.4.1 show

Show a node definition. If the node parameter is omitted then all nodes are shown.

Usage:

```
show [<node>]
```

3.4.2 standby/online

Set a node to standby or online status. The node parameter defaults to the node where the command is run.

Usage:

```
standby [<node>]
online [<node>]
```

3.4.3 delete

Delete a node. This command will remove the node from the CIB and, in case the heartbeat stack is running, run hb_delnode too.

Usage:

```
delete <node>
```

3.4.4 attribute

Edit node attributes. This kind of attribute should refer to relatively static properties, such as memory size.

Usage:

```
attribute <node> set <attr> <value>
attribute <node> delete <attr>
attribute <node> show <attr>
```

Example:

```
attribute node_1 set memory_size 4096
```

3.4.5 status-attr

Edit node attributes which are in the CIB status section, i.e. attributes which hold properties of a more volatile nature. One typical example is attribute generated by the `pingd` utility.

Usage:

```
status-attr <node> set <attr> <value>
status-attr <node> delete <attr>
status-attr <node> show <attr>
```

Example:

```
status-attr node_1 show pingd
```

3.5 options

The user may set various options for the CLI program itself.

3.5.1 skill-level

Based on the skill-level setting, the user is allowed to use only a subset of commands. There are three levels: operator, administrator, and expert. The operator level allows only commands at the `resource` and `node` levels, but not editing or deleting resources. The administrator may do that and may also configure the cluster at the `configure` level and manage the shadow CIBs. The expert may do all.

Usage:

```
skill-level level

level :: operator | administrator | expert
```

3.5.2 user

Sufficient privileges are necessary in order to manage a cluster: programs such as `crm_verify` or `crm_resource` and, ultimately, `cibadmin` have to be run either as `root` or as the CRM owner user (typically `hacluster`). You don't have to worry about that if you run `crm` as `root`. A more secure way is to run the program with your usual privileges, set this option to the appropriate user (such as `hacluster`), and setup the `sudoers` file.

Usage:

```
user system-user
```

Example:

```
user hacluster
```

3.5.3 editor

The `edit` command invokes an editor. Use this to specify your prefered editor program. If not set, it will default to either the value of the `EDITOR` environment variable or to one of the standard UNIX editors (`vi`,`emacs`,`nano`).

Usage:

```
editor program
```

Example:

```
editor vim
```

3.5.4 pager

The `view` command displays text through a pager. Use this to specify your prefered pager program. If not set, it will default to either the value of the `PAGER` environment variable or to one of the standard UNIX system pagers (`less`,`more`,`pg`).

3.5.5 output

`crm` can adorn configurations in two ways: in color (similar to for instance the `ls -color` command) and by showing keywords in upper case. Possible values are `plain`, `color`, and *uppercase*. It is possible to combine the latter two in order to get an upper case xmass tree. Just set this option to `color,uppercase`.

3.5.6 colorscheme

With `output` set to `color`, a comma separated list of colors from this option are used to emphasize:

- keywords
- object ids
- attribute names
- attribute values
- scores
- resource references

`crm` can show colors only if there is curses support for python installed (usually provided by the `python-curses` package). The colors are whatever is available in your terminal. Use `normal` if you want to keep the default foreground color.

This user preference defaults to `yellow,normal,cyan,red,green,magenta` which is good for terminals with dark background. You may want to change the color scheme and save it in the preferences file for other color setups.

Example:

```
colorscheme yellow,normal,blue,red,green,magenta
```

3.5.7 check-frequency

Semantic check of the CIB or elements modified or created may be done on every configuration change (`always`), when verifying (`on-verify`) or `never`. It is by default set to `always`. Experts may want to change the setting to `on-verify`.

The checks require that resource agents are present. If they are not installed at the configuration time set this preference to `never`.

See Configuration semantic checks for more details.

3.5.8 `check-mode`

Semantic check of the CIB or elements modified or created may be done in the `strict` mode or in the `relaxed` mode. In the former certain problems are treated as configuration errors. In the `relaxed` mode all are treated as warnings. The default is `strict`.

See Configuration semantic checks for more details.

3.5.9 `show`

Display all current settings.

3.5.10 `save`

Save current settings to the rc file (`$HOME/.crm.rc`). On further `crm` runs, the rc file is automatically read and parsed.

3.6 `configure`

This level enables all CIB object definition commands.

The configuration may be logically divided into four parts: nodes, resources, constraints, and (cluster) properties and attributes. Each of these commands support one or more basic CIB objects.

Nodes and attributes describing nodes are managed using the `node` command.

Commands for resources are:

- `primitive`
- `monitor`
- `group`
- `clone`
- `ms`/`master` (master-slave)

There are three types of constraints:

- `location`
- `colocation`
- `order`

Finally, there are the cluster properties, resource meta attributes defaults, and operations defaults. All are just a set of attributes. These attributes are managed by the following commands:

- `property`
- `rsc_defaults`

- op_defaults

The changes applied to the current CIB only on ending the configuration session or using the commit command.

3.6.1 node

The node command describes a cluster node. Nodes in the CIB are commonly created automatically by the CRM. Hence, you should not need to deal with nodes unless you also want to define node attributes. Note that it is also possible to manage node attributes at the node level.

Usage:

```
node <uname>[:<type>]
  [attributes <param>=<value> [<param>=<value>...]]

type :: normal | member | ping
```

Example:

```
node node1
node big_node attributes memory=64
```

3.6.2 primitive

The primitive command describes a resource. It may be referenced only once in group, clone, or master-slave objects. If it's not referenced, then it is placed as a single resource in the CIB.

Operations may be specified in three ways. "Anonymous" as a simple list of "op" specifications. Use that if you don't want to reference the set of operations elsewhere. That's by far the most common way to define operations. If reusing operation sets is desired, use the "operations" keyword along with the id to give the operations set a name and the id-ref to reference another set of operations.

Operation's attributes which are not recognized are saved as instance attributes of that operation. A typical example is OCF_CHECK_LEVEL.

Usage:

```
primitive <rsc> [<class>:[<provider>:]]<type>
  [params attr_list]
  [meta attr_list]
  [operations id_spec]
    [op op_type [<attribute>=<value>...] ...]
```

```
attr_list :: [$id=<id>] <attr>=<val> [<attr>=<val>...] | $id-ref=<id>
id_spec :: $id=<id> | $id-ref=<id>
op_type :: start | stop | monitor
```

Example:

```
primitive apcfence stonith:apcsmart \
  params ttydev=/dev/ttyS0 hostlist="node1 node2" \
  op start timeout=60s \
  op monitor interval=30m timeout=60s

primitive www8 apache \
  params configfile=/etc/apache/www8.conf \
  operations $id-ref=apache_ops

primitive db0 mysql \
  params config=/etc/mysql/db0.conf \
  op monitor interval=60s \
  op monitor interval=300s OCF_CHECK_LEVEL=10
```

3.6.3 `monitor`

Monitor is by far the most common operation. It is possible to add it without editing the whole resource. Also, long primitive definitions may be a bit uncluttered. In order to make this command as concise as possible, less common operation attributes are not available. If you need them, then use the `op` part of the `primitive` command.

Usage:

```
monitor <rsc>[:<role>] <interval>[:<timeout>]
```

Example:

```
monitor apcfence 60m:60s
```

Note that after executing the command, the monitor operation may be shown as part of the primitive definition.

3.6.4 `group`

The `group` command creates a group of resources.

Usage:

```
group <name> <rsc> [<rsc>...]
  [meta attr_list]
  [params attr_list]

attr_list :: [$id=<id>] <attr>=<val> [<attr>=<val>...] | $id-ref=<id>
```

Example:

```
group internal_www disk0 fs0 internal_ip apache \
  meta target_role=stopped
```

3.6.5 clone

The clone command creates a resource clone. It may contain a single primitive resource or one group of resources.

Usage:

```
clone <name> <rsc>
  [meta attr_list]
  [params attr_list]

attr_list :: [$id=<id>] <attr>=<val> [<attr>=<val>...] | $id-ref=<id>
```

Example:

```
clone cl_fence apc_1 \
  meta clone-node-max=1 globally-unique=false
```

3.6.6 ms (master)

The ms command creates a master/slave resource type. It may contain a single primitive resource or one group of resources.

Usage:

```
ms <name> <rsc>
  [meta attr_list]
  [params attr_list]

attr_list :: [$id=<id>] <attr>=<val> [<attr>=<val>...] | $id-ref=<id>
```

Example:

```
ms disk1 drbd1 \
  meta notify=true globally-unique=false
```

Note on `id-ref` usage

Instance or meta attributes (‘`params` and `meta`) may contain a reference to another set of attributes. In that case, no other attributes are allowed. Since attribute sets” ids, though they do exist, are not shown in the `crm`, it is also possible to reference an object instead of an attribute set. `crm` will automatically replace such a reference with the right id:

```
crm(live)configure# primitive a2 www-2 meta $id-ref=a1
crm(live)configure# show a2
primitive a2 ocf:heartbeat:apache \
    meta $id-ref="a1-meta_attributes"
    [...]
```

It is advisable to give meaningful names to attribute sets which are going to be referenced.

3.6.7 `location`

`location` defines the preference of nodes for the given resource. The location constraints consist of one or more rules which specify a score to be awarded if the rule matches.

Usage:

```
location <id> <rsc> {node_pref|rules}

node_pref :: <score>: <node>

rules ::
  rule [id_spec] [$role=<role>] <score>: <expression>
  [rule [id_spec] [$role=<role>] <score>: <expression> ...]

id_spec :: $id=<id> | $id-ref=<id>
score :: <number> | <attribute> | [-]inf
expression :: <simple_exp> [bool_op <simple_exp> ...]
bool_op :: or | and
simple_exp :: <attribute> [type:]<binary_op> <value>
              | <unary_op> <attribute>
              | date <date_expr>
type :: string | version | number
binary_op :: lt | gt | lte | gte | eq | ne
unary_op :: defined | not_defined

date_expr :: lt <end>
```

```
                | gt <start>
                | in_range start=<start> end=<end>
                | in_range start=<start> <duration>
                | date_spec <date_spec>
duration|date_spec ::
               hours=<value>
               | monthdays=<value>
               | weekdays=<value>
               | yearsdays=<value>
               | months=<value>
               | weeks=<value>
               | years=<value>
               | weekyears=<value>
               | moon=<value>
```

Examples:

```
location conn_1 internal_www 100: node1

location conn_1 internal_www \
  rule 50: #uname eq node1 \
  rule pingd: defined pingd

location conn_2 dummy_float \
  rule -inf: not_defined pingd or pingd lte 0
```

3.6.8 colocation (collocation)

This constraint expresses the placement relation between two resources.

Usage:

```
colocation <id> <score>: <rsc>[:<role>] <rsc>[:<role>]
```

Example:

```
colocation dummy_and_apache -inf: apache dummy
```

3.6.9 order

This constraint expresses the order of actions on two resources.

Usage:

```
order <id> score-type: <first-rsc>[:<action>] <then-rsc>[:<action>]
  [symmetrical=<bool>]

score-type :: advisory | mandatory | <score>
```

Example:

```
order c_apache_1 mandatory: apache:start ip_1
```

3.6.10 property

Set the cluster (crm_config) options.

Usage:

```
property [$id=<set_id>] <option>=<value> [<option>=<value> ...]
```

Example:

```
property stonith-enabled=true
```

3.6.11 rsc_defaults

Set defaults for the resource meta attributes.

Usage:

```
rsc_defaults [$id=<set_id>] <option>=<value> [<option>=<value> ...]
```

Example:

```
rsc_defaults failure-timeout=3m
```

3.6.12 op_defaults

Set defaults for the operations meta attributes.

Usage:

```
op_defaults [$id=<set_id>] <option>=<value> [<option>=<value> ...]
```

Example:

```
op_defaults record-pending=true
```

3.7 `show`

The `show` command displays objects. It may display all objects or a set of objects. The user may also choose to see only objects which were changed. Optionally, the XML code may be displayed instead of the CLI representation.

Usage:

```
show [xml] [<id> ...]
show [xml] changed
```

3.8 `edit`

This command invokes the editor with the object description. As with the `show` command, the user may choose to edit all objects or a set of objects.

If the user insists, he or she may edit the XML edition of the object. If you do that, don't modify any id attributes.

Usage:

```
edit [xml] [<id> ...]
edit [xml] changed
```

3.9 `delete`

Delete one or more objects. If an object to be deleted belongs to a container object, such as a group, and it is the only resource in that container, then the container is deleted as well. Any related constraints are removed as well.

Usage:

```
delete <id> [<id>...]
```

3.10 `rename`

Rename an object. It is recommended to use this command to rename a resource, because it will take care of updating all related constraints and a parent resource. Changing ids with the edit command won't have the same effect.

If you want to rename a resource, it must be in the stopped state.

Usage:

```
rename <old_id> <new_id>
```

3.11 refresh

Refresh the internal structures from the CIB. All changes made during this session are lost.

Usage:

```
refresh
```

3.12 erase

The `erase` clears all configuration. Apart from nodes. To remove nodes, you have to specify an additional keyword `nodes`.

Note that removing nodes from the live cluster may have some strange/interesting/unwelcome effects.

Usage:

```
erase [nodes]
```

3.13 ptest

Show PE (policy engine) motions using ptest.

A CIB is constructed using the current user edited version and the status from the running CIB. The resulting CIB is run through ptest to show changes. If you have graphviz installed and X11 session, dotty(1) is run to display the changes graphically.

Usage:

```
ptest
```

3.14 commit

Commit the current configuration to the CIB in use. As noted elsewhere, commands in a configure session don't have immediate effect on the CIB. All changes are applied at one point in time, either using `commit` or when the user leaves the configure level. In case the CIB in use changed in the meantime, presumably by somebody else, the CLI will refuse to apply the changes. If you know that it's fine to still apply them add `force`.

Usage:

```
commit [force]
```

3.15 upgrade

If you get the `CIB not supported` error, which typically means that the current CIB version is coming from the older release, you may try to upgrade it to the latest revision. The command to perform the upgrade is:

```
# cibadmin --upgrade --force
```

If we don't recognize the current CIB as the old one, but you're sure that it is, you may force the command.

Usage:

```
upgrade [force]
```

3.16 verify

Verify the contents of the CIB which would be committed.

Usage:

```
verify
```

3.17 save

Save the configuration of the current level to a file. Optionally, as XML.

Usage:

```
save [xml] <file>
```

Example:

```
save myfirstcib.txt
```

3.18 load

Load a part of configuration (or all of it) from a local file or a network URL. The `replace` method replaces the current configuration with the one from the source. The `update` tries to import the contents into the current configuration. The file may be a CLI file or an XML file.

Usage:

```
load [xml] method URL

method :: replace | update
```

Example:

```
load xml update myfirstcib.xml
load xml replace http://storage.big.com/cibs/bigcib.xml
```

3.19 template

User may be assisted in the cluster configuration by templates prepared in advance. Templates consist of a typical ready configuration which may be edited to suit particular user needs.

This command enters a template level where additional commands for configuration/template management are available.

3.19.1 new

Create a new configuration from one or more templates. Note that configurations and templates are kept in different places, so it is possible to have a configuration name equal a template name.

Usage:

```
new <config> <template> [<template> ...]
```

Example:

```
new vip virtual-ip
```

3.19.2 load

Load an existing configuration. Further `edit`, `show`, and `apply` commands will refer to this configuration.

Usage:

```
load <config>
```

3.19.3 edit

Edit current or given configuration using your favourite editor.

Usage:

```
edit [<config>]
```

3.19.4 `delete`

Remove a configuration. The loaded (active) configuration may be removed by force.

Usage:

```
delete <config> [force]
```

3.19.5 `list`

List existing configurations or templates.

Usage:

```
list [templates]
```

3.19.6 `apply`

Copy the current or given configuration to the current CIB. The CIB is effectively replaced.

Usage:

```
apply [<config>]
```

3.19.7 `show`

Process the current or given configuration and display the result.

Usage:

```
show [<config>]
```

3.20 `end` (`cd`, `up`)

The `end` command ends the current level and the user moves to the parent level. This command is available everywhere.

Usage:

```
end
```

3.21 help

The `help` command prints help for the current level or for the specified topic (command). This command is available everywhere.

Usage:

```
help [<topic>]
```

3.22 quit (exit, bye)

Leave the program.