Bacula Enterprise User Interfaces: Console

Bacula Systems Documentation

Contents

1 Console Installation **2**
1.1 Console Configuration 2
1.2 Console Management 2

Contents

To allow interaction from administrators or users, Bacula uses *Consoles*. The Bacula Console is a program that allows the user or the System Administrator to interact with the Bacula Director Daemon while the daemon is running. Note that, even when managing storage or checking client status, the Console interacts with the Director only, which in turn contacts the other daemons as needed.

Since the Console program interacts with the Director through the network, the Console and Director programs do not necessarily need to run on the same machine. In fact, in an installation containing a single tape drive, a certain minimal knowledge of the Console program may be needed in order for Bacula to be able to write on more than one Volume, because when Bacula requests a new one, it waits until the user, via the Console program, indicates that the new Volume is mounted or labeled to be used.

1 Console Installation

The Console is installed by default with the Director or the File Daemon installation.

The Console can be installed also on its own with the package named *bacula-enterprise-console*.

1.1 Console Configuration

When the *Console* starts, it reads a standard **Bacula** configuration file named **bconsole.conf** or **bat.conf** in the case of the *BAT* QT *Console* version from the current directory unless you specify the **-c** command line option (see below). This file allows default configuration of the *Console*, and at the current time, the only Resource Record defined is the **Director**, which gives the *Console* the name and address of the *Director*.

1.2 Console Management

Running the Console Program

Plese type *sudo -u bacula /opt/bacula/bin/bconsole* in order to run the console.

The Bacula **Console** program can be run with the following options:

```
Usage: bconsole [-s] [-c config-file] [-d debug-level]
       -D <dir>    select a Director
       -l          list Directors defined
       -L          list Consoles defined
       -C <cons>   select a console
```

(continues on next page)

(continued from previous page)

```
      -c <file>   set configuration file to file
      -d <nn>     set debug level to <nn>
      -dt         print timestamp in debug output
      -n          no conio
      -s          no signals
      -u <nn>     set command execution timeout to <nn> seconds
      -t          test - read configuration and exit
      -?          print this message.
```

After launching the *Console* program (**bconsole**), it will prompt you for the next command with an asterisk (*). Generally, for all commands, you can simply enter the command name and the Console program will prompt you for the necessary arguments. Alternatively, in most cases, you may enter the command followed by arguments. The general format is:

```
<command> <keyword1>[=<argument1>] <keyword2>[=<argument2>] ...
```

where **command** is one of the commands listed below; **keyword** is one of the keywords listed below (usually followed by an argument); and **argument** is the value. The command may be abbreviated to the shortest unique form. If two commands have the same starting letters, the one that will be selected is the one that appears first in the **help** listing. If you want the second command, simply spell out the full command. None of the keywords following the command may be abbreviated.

For example:

```
list files jobid=23
```

will list all files saved for JobId 23. Or:

```
show pools
```

will display all the **Pool** resource records.

An abbreviated command would be similar to

```
s dir
```

which will show the overall status of the Director currently connected to.

In many cases, **bconsole** will provide tab completion of commands, options names and arguments. Also, command line history and editing may be available.

Depending on how the program is built, the regular **readline** key bindings will be used. If the readline functionality is used, and not the alternative Bacula native one, the full functionality of **readline**, including command history and configuration through the **~/.inputrc** file, is available. We recommend to look at the available configuration as provided by the operating system distribution.

Tab completion is a particularly useful tool, but depends on availability of information that is gathered from the Director at run time, using some dot commands behind the scenes. Thus, it depends on the availability of the commands needed, which may be restricted in some cases. Please see *Special dot Commands* for more details.

Command line completion can simplify interactive activity a lot:

```
~$ bconsole -c .config/bacula.org/k8s-bconsole.conf
Connecting to Director am-d11-director-12-8-3-tst.supportlab.lan:9101
1000 OK: 10002 am-d11-director-12-8-3-tst-dir Version: 12.8.3 (28 September␣
```

(continues on next page)

(continued from previous page)

```
↪2021)
Enter a period to cancel a command.
*st<tab><tab>
statistics  status      stop
*status cli<tab>ent=<tab>am-u20-k8s-master01-bck-fd<enter>
Connecting to Client am-u20-k8s-master01-bck-fd at am-u20-k8s-master01-bck.
↪supportlab.lan:9102

am-u20-k8s-master01-bck-fd Version: 12.8.3 (28 September 2021) ...
```

The maximum command line length is limited to **511** characters, so if you are scripting the console, you may need to take some care to limit the line length.

Stopping the Console Program

Normally, you simply enter **quit** or **exit** and the *Console* program will terminate. However, it waits until the *Director* acknowledges the command. If the *Director* is already doing a lengthy command (e.g. **prune**), it may take some time. If you want to immediately terminate the *Console* program, enter the **.quit** command.

There is currently no way to interrupt a *Console* command once issued (i.e. `Ctrl-C` does not work). However, if you are at a prompt that is asking you to select one of several possibilities and you would like to abort the command, you can enter a period (**.**), and in most cases, you will either be returned to the main command prompt or if appropriate the previous prompt (in the case of nested prompts). In a few places such as where it is asking for a Volume name, the period will be taken to be the Volume name. In that case, you will most likely be able to cancel at the next prompt.

Alphabetic List of Console Keywords

Unless otherwise specified, each of the following keywords takes an argument, which is specified after the keyword following an equal sign. For example:

```
jobid=536
```

all
: Permitted on the **status** and **show** commands to specify all components or resources respectively. Takes no argument.

frompool
: Permitted on the **update volume** command to specify that the volume specified on the command line should be updated with pool parameters.

allfrompool=<pool>
: Permitted on the **update** command to specify that all Volumes in the pool (specified on the command line) should be updated.

fromallpools
: Permitted on the **update** command to specify that all Volumes in all pools should be updated.

before
: Used in the **restore** command.

bootstrap
: Used in the **restore** command.

catalog
Allowed in the **use** command to specify the catalog name to be used.

catalogs
Used in the **show** command. Takes no arguments.

client | fd

clients
Used in the **show**, **list**, and **llist** commands. Takes no arguments.

counters
Used in the **show** command. Takes no arguments.

current
Used in the **restore** command. Takes no argument.

days
Used to define the number of days the **list nextvol** command should consider when looking for jobs to be run. The **days** keyword can also be used on the **status dir** command so that it will display jobs scheduled for the number of days you want.

devices
Used in the **show** command. Takes no arguments.

dir | director

directors
Used in the **show** command. Takes no arguments.

directory
Used in the **restore** command. Its argument specifies the directory to be restored.

enabled
This keyword can appear on the **update volume** as well as the **update slots** commands, and can allows one of the following arguments: **yes**, **true**, **no**, **false**, **archived**, **0**, **1**, **2**. Where **0** corresponds to no or false, **1** corresponds to **yes** or **true**, and **2** corresponds to **archived**. Archived volumes will not be used, nor will the Media record in the catalog be pruned. Volumes that are not enabled, will not be used for backup or restore.

done
Used in the **restore** command. Takes no argument.

file
Used in the **restore** command.

files
Used in the **list** and **llist** commands. Takes no arguments.

fileset

filesets
Used in the **show** command. Takes no arguments.

help
Used in the **show** command. Takes no arguments.

jobs
Used in the **show**, **list** and **llist** commands. Takes no arguments.

jobmedia
Used in the **list** and **llist** commands. Takes no arguments.

jobtotals
Used in the **list** and **llist** commands. Takes no arguments.

jobid
The JobId is the numeric jobid that is printed in the Job Report output. It is the index of the database record for the given job. While it is unique for all the existing Job records in the catalog database, the same JobId can be reused once a Job is removed from the catalog. Probably you will refer specific Jobs that ran using their numeric JobId.

job | jobname
The **job** or **jobname** keyword refers to the name you specified in the **Job**, and hence it refers to any number of Jobs that ran. It is typically useful if you want to list all jobs of a particular name.

level

listing
Permitted on the **estimate** command. Takes no argument.

limit

messages
Used in the **show** command. Takes no arguments.

media
Used in the **list** and **llist** commands. Takes no arguments.

name
Used in the **list object** commands. Can specify an Object name.

nextvol | nextvolume
Used in the **list** and **llist** commands. Takes no arguments.

noautoparent
Used with the **restore** command. Takes no arguments.

object
Used in the **list** commands. Takes no arguments.

objectid
Used in the **list** and **restore** commands. Takes an ObjectId as argument.

order
Used in the **list** to sort records. Can take "ASC" or "DESC" as argument.

on
Takes no keyword.

off
Takes no keyword.

pool

pools
Used in the **show**, **list**, and **llist** commands. Takes no arguments.

select
Used in the **restore** command. Takes no argument.

limit
Used in the **setbandwidth** command. Takes integer in `KB/s` unit.

storages
Used in the **show** command. Takes no arguments.

schedules
Used in the **show** command. Takes no arguments.

sd | store | storage

tag
Used in the **list** command. Takes no arguments.

ujobid
The **ujobid** is a unique job identification that is printed in the Job Report output. At the current time, it consists of the Job name (from the **Name** for the job) appended with the date and time the job was run. This keyword is useful if you want to completely identify the Job instance run.

volume

volumes
Used in the **list** and **llist** commands. Takes no arguments.

where
Used in the **restore** command.

yes
Used in the **restore** command. Takes no argument.

Alphabetic List of Console Commands

The following commands are currently implemented:

add [pool=<pool-name> storage=<storage> jobid=<JobId>]
This command is used to add Volumes to an existing Pool. That is, it creates the Volume name in the catalog and inserts into the Pool in the catalog, but does not attempt to access the physical Volume. Once added, **Bacula** expects that Volume to exist and to be labeled. This command is not normally used since **Bacula** will automatically do the equivalent when Volumes are labeled. However, there may be times when you have removed a Volume from the catalog and want to later add it back.

Normally, the **label** command is used rather than this command because the **label** command labels the physical media (tape, disk, DVD, ...) and does the equivalent of the **add** command. The **add** command affects only the Catalog and not the physical media (data on Volumes). The physical media must exist and be labeled before use (usually with the **label** command). This command can, however, be useful if you wish to add a number of Volumes to the Pool that will be physically labeled at a later time. It can also be useful if you are importing a tape from another site. Please see the **label** command below for the list of legal characters in a Volume name.

autodisplay on/of
This command accepts **on** or **off** as an argument, and turns auto-display of messages on or off respectively. The default for the console program is **off**, which means that you will be notified when there are console messages pending, but they will not automatically be displayed.

When autodisplay is turned off, you must explicitly retrieve the messages with the **messages** command. When **autodisplay** is turned on, the messages will be displayed on the console as they are received.

automount on/of
This command accepts **on** or **off** as the argument, and turns auto-mounting of the Volume after a **label** command on or off respectively. The default is **on**. If **automount** is turned off, you must explicitly **mount** tape Volumes after a **label** command to use it.

cancel [jobid=<number> job=<job-name> ujobid=<unique-jobid>] | [inactive client=<client-name> storage=<storage-name>]

This command is used to cancel a job and accepts **jobid=nnn** or **job=xxx** as an argument where nnn is replaced by the JobId and xxx is replaced by the job name. If you do not specify a keyword, the *Console* program will prompt you with the names of all the active jobs allowing you to choose one.

Once a Job is marked to be canceled, it may take a bit of time (generally within a minute but up to two hours) before the Job actually terminates, depending on what operations it is doing. Don't be surprised that you receive a Job not found message. That just means that one of the three daemons had already canceled the job. Messages numbered in the 1000's are from the Director, 2000's are from the File daemon and 3000's from the Storage daemon.

If the Job is still active on the Storage Daemon and/or the File Daemon, but not on the Director, it is possible to stop the Job with the **inactive** option of the **cancel** command.

cloud [storage=<sd-name> volume=<vol-name> allpools allfrompool pool=<p-name> mediatype=<type-name> drive=<num> slots=<num>] [truncate | prune | list | upload]

The **cloud bconsole** command allows you to do a number of things with cloud volumes.

The **truncate** option of the cloud command will attempt to truncate the local cache for the specified Volume.

The **prune** option of the cloud command will attempt to prune the local cache for the specified Volume. **Bacula** will respect the **CacheRetention** volume attribute to determine if the cache can be truncated or not. Only parts that are uploaded to the cloud will be deleted from the cache.

The **upload** option of the cloud command will attempt to upload the specified Volumes.

The **list** option of the cloud command will list volumes stored in the Cloud. If a volume name is specified, the command will list all parts for the given volume.

create [pool=<pool-name>]

This command is not normally used as the Pool records are automatically created by the *Director* when it starts based on what it finds in the conf file. If needed, this command can be to create a *Pool* record in the database using the **Pool** record defined in the *Director*'s configuration file. So in a sense, this command simply transfers the information from the **Pool** in the configuration file into the *Catalog*. Normally this command is done automatically for you when the *Director* starts providing the *Pool* is referenced within a **Job**. If you use this command on an existing *Pool*, it will automatically update the *Catalog* to have the same information as the **Pool**. After creating a *Pool*, you will most likely use the **label** command to label one or more volumes and add their names to the Media database.

When starting a *Job*, if **Bacula** determines that there is no *Pool* record in the database, but there is a **Pool** of the appropriate name, it will create it for you. If you want the *Pool* record to appear in the database immediately, simply use this command to force it to be created.

dedup

This command is used to manage Global Deduplication Engine.

[storage=<storage-name> vacuum [forceoptimize] [holepunching] [check-miss] [checkindex] usage scrub [run | stop | suspend | resume | status] [worker=<nn>] [reset] tune indexmemory bnum_min=<nn> bnum_max=<nn> mlock_strategy=<nn> mlock_max=<nn> hole_size=<nn> rehydra_check_hash=<nn> scrub_bwlimit=<nn> qindex [[:NN:] [:XXXXXXXX] [:X{32}] [0xXX] [all]] qchunk [[size] [check-hash] [:NN:] [0xXX] [all]] qextent [[:NN:] [0xXX] all] qcontainer]

delete [volume=<vol-name> pool=<pool-name> job jobid=<id> object [objectid=id]]

The **delete** command is used to delete a *Volume*, *Pool* , Object, Snapshot or *Client* record from

the *Catalog* as well as all associated catalog Volume records that were created. This command operates only on the *Catalog* database and has no effect on the actual data written to a *Volume*. This command can be dangerous and we strongly recommend that you do not use it unless you know what you are doing.

If the keyword **Volume** appears on the command line, the named *Volume* will be deleted from the *Catalog*, if the keyword **Pool** appears on the command line, a *Pool* will be deleted, if the keyword **Object** appears on the command line, an Object and all of its associated records (File) will be deleted from the catalog, and if the keyword **Job** appears on the command line, a *Job* and all of its associated records (File and JobMedia) will be deleted from the catalog. The full form of this command is:

```
delete pool=<pool-name>
```

or

```
delete volume=<volume-name> pool=<pool-name>
```

or

```
delete JobId=<job-id> JobId=<job-id2> ...
```

or

```
delete JobId=n,m,o-r,t ...
```

or

```
delete client=<client-name>
```

or

```
delete snapshot
```

or

```
delete object [objectid=id [all]] [category=obj_cat] [type=obj_type]␣
↪[name=obj_name] [uuid=obj_uuid] [source=obj_source]
```

The first form deletes a *Pool* record from the catalog database. The second form deletes a *Volume* record from the specified pool in the *Catalog* database. The third form deletes the specified record from the catalog database. The fourth form deletes records for JobIds **n, m, o, p, q, r** and **t**. Where each one of the **n,m,...** is, of course, a number. That is a **delete jobid** accepts lists and ranges of jobids.

The deletion of a *Client* record will prune all Job records associated with the Client. This command is possible only once the *Client* resource of the given name is no longer defined in the Director configuration file.

The deletion of a Snapshot record can be done with the sixth command.

The deletion of an Object records can be done with the last command. Different filters can be used along with 'delete object' command: category, type, name, uuid or source. Passing specific object id can be also used as a filter to delete many objects at once. Following command will delete all objects records with the same category, type, name, uuid and source as object with given id.

```
delete object objectid=1 all
```


disable job<job-name>
This command permits you to disable a for automatic scheduling. The job may have been previously enabled with the **Job Enabled** or using the console **enable** command. The next time the *Director* is restarted, the **Enable/Disable** state will be set to the value in the **Job** (default **enabled**) as defined in the **bacula-dir.conf** file.

disable jobs all
This command permits you to disable all Jobs for automatic scheduling. The next time the *Director* is restarted, the **Enable/Disable** state will be set to the value in the *Job* resource (default **enabled**) as defined in the **bacula-dir.conf** file.

enable job<job-name>
This command permits you to enable a for automatic scheduling. The job may have been previously disabled with the *Job* resource **Enabled** or using the console **disable** command. The next time the *Director* is restarted, the **Enable/Disable** state will be set to the value in the **Job** (default **enabled**) as defined in the **bacula-dir.conf** file.

enable jobs all
This command permits you to enable all Jobs for automatic scheduling. It does not enable jobs which have **Disabled** directive. The next time the *Director* is restarted, the **Enable/Disable** state will be set to the value in the **Job** (default **enabled**) as defined in the **bacula-dir.conf** file.

estimate
Using this command, you can get an idea how many files will be backed up, or if you are unsure about your Include statements in your *Fileset*, you can test them without doing an actual backup. The default is to assume a Full backup. However, you can override this by specifying a **level=Incremental** or **level=Differential** on the command line. A name must be specified or you will be prompted for one, and optionally a *Client* and *Fileset* may be specified on the command line. It then contacts the client which computes the number of files and bytes that would be backed up. Please note that this is an estimate calculated from the number of blocks in the file rather than by reading the actual bytes. As such, the estimated backup size will generally be larger than an actual backup.

The **estimate** command can use the accurate code to detect changes and give a better estimation. You can set the accurate behavior on command line using **accurate=yes/no** or use the setting as default value.

Optionally you may specify the keyword **listing** in which case, all the files to be backed up will be listed. Note, it could take quite some time to display them if the backup is large. The full form is:

```
estimate job=<job-name> listing client=<client-name> accurate=<yes/no>␣
↪fileset=<fileset-name> level=<level-name>
```

Specification of the **job** is sufficient, but you can also override the **client**, **fileset**, **accurate** and/or **level** by specifying them on the **estimate** command line.

As an example, you might do:

```
@tall /tmp/listing
estimate job=NightlySave listing level=Incremental
@tall
```

which will do a full listing of all files to be backed up for the Job **NightlySave** during an Incremental save and put it in the file **/tmp/listing**. Note, the byte estimate provided by this command is based on the file size contained in the directory item. This can give wildly incorrect estimates of the actual storage used if there are sparse files on your systems. Sparse files are often found on 64 bit systems for certain system files. The size that is returned is the size **Bacula** will backup if the

sparse option is not specified in the *Fileset*. There is currently no way to get an estimate of the real file size that would be found should the sparse option be enabled.

exit

This command terminates the *Console* program.

gui

Invoke the non-interactive gui mode.

```
gui [on|off]
```

help

This command displays the list of commands available.

label

This command is used to label physical volumes. The full form of this command is:

```
label storage=<storage-name> volume=<volume-name>
slot=<slot>
```

If you leave out any part, you will be prompted for it. The media type is automatically taken from the **Storage** definition that you supply. Once the necessary information is obtained, the *Console* program contacts the specified daemon and requests that the *Volume* be labeled. If the Volume labeling is successful, the Console program will create a *Volume* record in the appropriate *Pool*.

The Volume name is restricted to letters, numbers, and the special characters hyphen (-), underscore (_), colon (:), and period (.). All other characters including a space are invalid. This restriction is to ensure good readability of Volume names to reduce operator errors.

Please note, when labeling a blank tape, **Bacula** will get **read I/O error** when it attempts to ensure that the tape is not already labeled. If you wish to avoid getting these messages, please write an EOF mark on your tape before attempting to label it:

```
mt rewind
mt weof
```

The **label** command can fail for a number of reasons:

1. The *Volume* name you specify is already in the Volume database.
2. The daemon has a tape or other Volume already mounted on the device, in which case you must **unmount** the device, insert a blank tape, then do the **label** command.
3. The Volume in the device is already a **Bacula** labeled Volume. (**Bacula** will never relabel a **Bacula** labeled Volume unless it is recycled and you use the **relabel** command).
4. There is no Volume in the drive.

There are two ways to relabel a volume that already has a **Bacula** label. The brute force method is to write an end of file mark on the tape using the system **mt** program, something like the following:

```
mt -f /dev/st0 rewind
mt -f /dev/st0 weof
```

For a disk volume, you would manually delete the Volume.

Then you use the **label** command to add a new label. However, this could leave traces of the old volume in the catalog.

The preferable method to relabel a Volume is to first **purge** the volume, either automatically, or explicitly with the **purge** command, then use the **relabel** command described below.

If your autochanger has barcode labels, you can label all the Volumes in your autochanger one after another by using the **label barcodes** command. For each tape in the changer containing a barcode, **Bacula** will mount the tape and then label it with the same name as the barcode. An appropriate Media record will also be created in the catalog. Any barcode that begins with the same characters as specified on the " **Cleanprefix = xxx** " directive in the *Director*'s *Pool* resource, will be treated as a cleaning tape, and will not be labeled. However, an entry for the cleaning tape will be created in the catalog. For example with:

```
Pool {
  Name ...
  Cleaning Prefix = "CLN"
}
```

Any slot containing a barcode of `CLNxxxx` will be treated as a cleaning tape and will not be mounted. Note, the full form of the command is:

```
label storage=xxx pool=yyy slots=1-5,10 barcodes
```

list

The **list** command lists the requested contents of the Catalog. The various fields of each record are listed on a single line. To see the complete list of options, use the `help list` command. The various forms of the list command are:

```
list events

list jobs

list jobid=<id>    (list jobid id)

list jobs joberrors   (list jobs with errors)

list jobs jobstatus=f     (list jobs with jobstatus f)

list jobs limit=10 order=desc   (list the last 10 jobs)

list jobs limit=10 order=asc   (list the first 10 jobs)

list jobs client=xxx     (list jobs for client xxx)

list copies   (list copy jobs)

list ujobid=<unique job name>   (list job with unique name)

list job=<job-name>   (list all jobs with "job-name")

list jobname=<job-name>   (same as above)

  In the above, you can add "limit=nn" to limit the output to nn jobs.

list joblog jobid=<id>     (list job output if recorded in the catalog)
```

(continues on next page)

(continued from previous page)

```
list jobmedia

list jobmedia jobid=<id>

list jobmedia job=<job-name>

list fileevents jobid=<xx>   (list errors associated with a specific␣
↪JobId)

list files jobid=<id>

list files job=<job-name>

  In the above, you can add type=<all|deleted> to see all file records␣
↪or only deleted records.

list files type=<deleted|all> jobid=<id>

list pools

list clients

list jobtotals

list metadata type=email from=<str> cc=<str> subject=<str> client=<cli>␣
↪limit=<nb>

list metadata type=attachment id=<str>

list objects (list plugin objects)

list objects jobid=<id>

list object client=<client> (list plugin objects of specified client)

list object type=<type> (list plugin objects of specified type)

list objects category=<category> (list plugin objects of specified␣
↪category)

list objects limit=10 order=desc (list the last 10 objects)

list objects limit=10 order=asc (list the first 10 objects)

list volumes

list volumes jobid=<id>

list volumes pool=<pool-name>

list volumes job=<job-name>
```

(continues on next page)

(continued from previous page)

```
list volume=<volume-name>

list nextvolume job=<job-name>

list nextvol job=<job-name>

list nextvol job=<job-name> days=nnn
```

What most of the above commands do should be more or less obvious. In general if you do not specify all the command line arguments, the command will prompt you for what is needed.

The **list nextvol** command will print the Volume name to be used by the specified job. You should be aware that exactly what Volume will be used depends on a lot of factors including the time and what a prior job will do. It may fill a tape that is not full when you issue this command. As a consequence, this command will give you a good estimate of what Volume will be used but not a definitive answer. In addition, this command may have certain side effect because it runs through the same algorithm as a job, which means it may automatically purge or recycle a Volume. By default, the job specified must run within the next two days or no volume will be found. You can, however, use the **days=nnn** specification to specify up to 50 days. For example, if on Friday, you want to see what Volume will be needed on Monday, for job MyJob, you would use **list nextvol job=MyJob days=3**.

If you wish to add specialized commands that list the contents of the catalog, you can do so by adding them to the **query.sql** file. However, this takes some knowledge of programming *SQL*. Please see the **query** command below for additional information. See below for listing the full contents of a catalog record with the **llist** command.

As an example, the command **list pools** might produce the following output:

```
+------+---------+---------+---------+----------+-------------+
| PoId | Name    | NumVols | MaxVols | PoolType | LabelFormat |
+------+---------+---------+---------+----------+-------------+
| 1    | Default | 0       | 0       | Backup   | *           |
| 2    | Recycle | 0       | 8       | Backup   | File        |
+------+---------+---------+---------+----------+-------------+
```

As mentioned above, the **list** command lists what is in the database. Some things are put into the database immediately when **Bacula** starts up, but in general, most things are put in only when they are first used, which is the case for a Client as with Job records, etc.

Bacula should create a client record in the database the first time you run a job for that client. Doing a **status** will not cause a database record to be created. The client database record will be created whether or not the job fails, but it must at least start. When the Client is actually contacted, additional info from the client will be added to the client record (a **uname -a** output).

If you want to see what *Client* resource you have available in your conf file, you use the Console command **show clients**.

llist

The **llist** or "long list" command takes all the same arguments that the **list** command described above does. The difference is that the **llist** command **list** the full contents of each database record selected. It does so by listing the various fields of the record vertically, with one field per line. It is possible to produce a very large number of output lines with this command.

If instead of the **list pools** as in the example above, you enter **llist pools** you might get the following output:


```
PoolId: 1
Name: Default
NumVols: 0
MaxVols: 0
UseOnce: 0
UseCatalog: 1
AcceptAnyVolume: 1
VolRetention: 1,296,000
VolUseDuration: 86,400
MaxVolJobs: 0
MaxVolBytes: 0
AutoPrune: 0
Recycle: 1
PoolType: Backup
LabelFormat: *

PoolId: 2
Name: Recycle
NumVols: 0
MaxVols: 8
UseOnce: 0
UseCatalog: 1
AcceptAnyVolume: 1
VolRetention: 3,600
VolUseDuration: 3,600
MaxVolJobs: 1
MaxVolBytes: 0
AutoPrune: 0
Recycle: 1
PoolType: Backup
LabelFormat: File
```

messages
This command causes any pending console messages to be immediately displayed.

memory
Print current memory usage.

mount
The **mount** command is used to get **Bacula** to read a volume on a physical device. It is a way to tell **Bacula** that you have mounted a tape and that **Bacula** should examine the tape. This command is normally used only after there was no Volume in a drive and **Bacula** requests you to mount a new Volume or when you have specifically unmounted a Volume with the **unmount** console command, which causes **Bacula** to close the drive. If you have an autoloader, the **mount** command will not cause **Bacula** to operate the autoloader unless you specify a **slot** and possibly a **drive**. The various forms of the mount command are:

```
mount storage=<storage-name> [ slot=<num> ] [ drive=<num> ]
```

```
mount [ jobid=<id> | job=<job-name> ]
```

If you have specified **Automatic Mount = yes** in the Storage daemon's **Device**, under most circumstances, **Bacula** will automatically access the Volume unless you have explicitly **unmount**ed it in the Console program.

prune

The **prune** command allows you to safely remove expired Jobs, Files and Statistics database records from the Catalog. This command works only on the Catalog database and does not affect data written to Volumes. In all cases, the **prune** command adheres to retention times of the specified records. You can prune expired File entries, Job records or statistics from the database. Additionally, when you prune volumes, you prune both expired Job and File records from Volumes, once all the records pertinent to the volume have been pruned, the volume status will change to **Purged**.

```
prune files|stats client=<client-name> [yes]

prune jobs (all|client=<client-name>) [yes]

prune volume(=<volume-name>| allfrompool (pool=<pool-name>|allpools))␣
↪[yes]

prune all
```

For a Volume to be pruned, the **VolStatus** must be *Full*, *Used*, or *Append*, otherwise the pruning will not take place.

When pruning Jobs With the **all** keyword, all combinations of **Client/Pool** present in the *Job* table will be pruned. **prune jobs all yes**

purge

The **purge** command will delete associated Catalog database records from Jobs and Volumes without considering the retention period. **purge** works only on the Catalog database and does not affect data written to Volumes. This command can be dangerous because you can delete catalog records associated with current backups of files, and we recommend that you do not use it unless you know what you are doing. The permitted forms of **purge** are:

```
purge files jobid=<jobid>|job=<job-name>|client=<client-name>
```

```
purge jobs client=<client-name> (of all jobs)
```

```
purge volume|volume=<vol-name> (of all jobs)
```

For the **purge** command to work on Volume Catalog database records the **VolStatus** must be **Append**, **Full**, **Used**, or **Error.**

The actual data written to the Volume will be unaffected by this command unless you are using the **ActionOnPurge=Truncate** option on those Media.

To ask **Bacula** to truncate your `Purged` volumes, you need to use the following command in interactive mode or in a **RunScript**:

```
* purge volume action=truncate storage=File allpools

# or by default, action=all
* purge volume action storage=File pool=Default
```

This is possible to specify the volume name, the media type, the pool, the storage, etc... (see **help purge**) Be sure that your storage device is idle when you decide to run this command.

query

This command reads a predefined SQL query from the query file (the name and location of the

query file is defined with the **QueryFile** record in the Director's configuration file). You are prompted to select a query from the file, and possibly enter one or more parameters, then the command is submitted to the Catalog database SQL engine.

The following queries are currently available (version 2.2.7):

```
Available queries:
1: List up to 20 places where a File is saved regardless of the directory
2: List where the most recent copies of a file are saved
3: List last 20 Full Backups for a Client
4: List all backups for a Client after a specified time
5: List all backups for a Client
6: List Volume Attributes for a selected Volume
7: List Volumes used by selected JobId
8: List Volumes to Restore All Files
9: List Pool Attributes for a selected Pool
10: List total files/bytes by Job
11: List total files/bytes by Volume
12: List Files for a selected JobId
13: List Jobs stored on a selected MediaId
14: List Jobs stored for a given Volume name
15: List Volumes Bacula thinks are in changer
16: List Volumes likely to need replacement from age or errors
Choose a query (1-16):
```

quit

This command terminates the console program. The console program sends the **quit** request to the Director and waits for acknowledgment. If the Director is busy doing a previous command for you that has not terminated, it may take some time. You may quit immediately by issuing the **.quit** command (i.e. quit preceded by a period).

relabel

This command is used to label physical volumes. The full form of this command is:

```
relabel storage=<storage-name> oldvolume=<old-volume-name> volume=
↪<newvolume-name> pool=<pool>
```

If you leave out any part, you will be prompted for it. In order for the Volume (old-volume-name) to be relabeled, it must be in the catalog, and the volume status must be marked **Purged** or **Recycle**. This happens automatically as a result of applying retention periods, or you may explicitly purge the volume using the **purge** command.

Once the volume is physically relabeled, the old data previously written on the Volume is lost and cannot be recovered.

release

This command is used to cause the Storage daemon to rewind (release) the current tape in the drive, and to re-read the Volume label the next time the tape is used.

```
release storage=<storage-name>
```

After a **release** command, the device is still kept open by **Bacula** (unless **Always Open** is set to **No** in the Storage Daemon's configuration) so it cannot be used by another program. However, with some tape drives, the operator can remove the current tape and to insert a different one, and when the next Job starts, **Bacula** will know to re-read the tape label to find out what tape is mounted. If

you want to be able to use the drive with another program (e.g. **mt**), you must use the **unmount** command to cause **Bacula** to completely release (close) the device.

reload

The **reload** command causes the Director to re-read its configuration file and apply the new values. The new values will take effect immediately for all new jobs. However, if you change schedules, be aware that the scheduler pre-schedules jobs up to two hours in advance, so any changes that are to take place during the next two hours may be delayed. Jobs that have already been scheduled to run (i.e. surpassed their requested start time) will continue with the old values. New jobs will use the new values. Each time you issue a **reload** command while jobs are running, the old config values will keptf until all jobs that were running before issuing the reload terminate, at which time the old config values will be released from memory. As a default a maximum number of 32 reload requests that can be made, which is generally sufficient. In the case that you make a very large number of reload requests, you may use the **Maximum Reload Requests** directive in the Director resource of **bacula-dir.conf** to set a larger maximum to that value you wish.

restart

The **restart** command allows console users to restart a canceled, failed, or incomplete Job. For canceled and failed Jobs, the Job will restart from the beginning. For incomplete Jobs the Job will restart at the point that it was stopped either by a stop command or by some recoverable failure.

The **restart** command, when entered in bconsole in this plain way, will create the following prompt:

```
*restart
You have the following choices:
    1: Incomplete
    2: Canceled
    3: Failed
    4: All
Select termination code: (1-4):
```

If you select the **All** option, you may see something like:

Select termination code: (1-4): 4

Table 1: bVerbatim

jobid	name	starttime	type	level	jobfiles	jobbytes	jobstatus
1	Incremental	2012-03-26 12:15:21	B	F	0	0	A
2	Incremental	2012-03-26 12:18:14	B	F	350	4,013,397	I
3	Incremental	2012-03-26 12:18:30	B	F	0	0	A
4	Incremental	2012-03-26 12:18:38	B	F	331	3,548,058	I

Enter the JobId list to select:

Then you may enter one or more JobIds to be restarted, which may take the form of a list of JobIds separated by commas, and/or JobId ranges such as **1-4**, which indicates you want to restart JobIds 1 through 4, inclusive.

The **restart** command accepts some parameters to simplify selection of the Job to restart, and to modify the behaviour of the restarted Job instance.

To Filter, you can use

- the keywords *failed*, *canceled* or *incomplete*
- the option *client=<name>*
- the option *job=<name>*

Note: The `restart` command has limitations with plugins, as it initiates the Job from scratch rather than continuing it. Bacula determines whether a Job is restarted or continued, but using the `restart` command will result in a new Job.

---

resume

The **resume** command does exactly the same thing as a **restart** command, but for some users the name may be more logical because in general the **restart** command is used to resume running a Job that was incomplete.

restore

The **restore** command allows you to select one or more Jobs (JobIds) to be restored using various methods. Once the JobIds are selected, the File records for those Jobs are placed in an internal **Bacula** directory tree, and the restore enters a file selection mode that allows you to interactively walk up and down the file tree selecting individual files to be restored.

```
restore storage=<storage-name> client=<backup-client-name> where=<path>
pool=<pool-name> fileset=<fileset-name> comment=<comment-specification>
restoreclient=<restore-client-name> objectid=<id> restorejob=<job-name>
jobuser=<user-name> jobgroup=<group-name>
noautoparent select current all done
```

Where **current**, if specified, tells the **restore** command to automatically select a restore to the most current backup. If not specified, you will be prompted. The **all** specification tells the **restore** command to restore all files. If it is not specified, you will be prompted for the files to restore.

The client keyword initially specifies the client from which the backup was made and the client to which the restore will be make. However, if the **restoreclient** keyword is specified, then the restore is written to that client.

The comment keyword can be used to store information related to the job in the catalog, like who has requested the job. It can have up to 4096 characters and cannot use the following characters : ‘ < > & \ “

The **noautoparent** keyword advises the restore preparation to not automatically select directory entries leading up to the explicitly marked files for restoration. Normally, such entries are selected and restored to ensure that proper permissions are recreated. However, in some cases this may not be needed or desirable, and thus, the functionality can be disabled here. Directories will still be created, but they will receive the default permissons; on Unix / Linux hosts, this is usually resulting in them being owned by root and having permissions set as defined by system defaults and `umask` modifier. This keyword is available as of Bacula Enterprise version 8.10.

The restore Job rarely needs to be specified, as bacula installations commonly only have a single restore job configured. However, for certain cases, such as a varying list of **RunScript** specifications, multiple restore jobs may be configured. The **restorejob** argument allows the selection of one of these jobs.

run

This command allows you to schedule jobs to be run immediately. The full form of the command is:

```
run job=<job-name> client=<client-name>
  fileset=<Fileset-name> level=<level-keyword>
  storage=<storage-name> where=<directory-prefix> comment=<comment-
↪specification>
  when=<universal-time-specification> spooldata=yes|no yes
```

The comment keyword can be used to store information related to the job in the catalog, like who has requested the job. It can have up to 4096 characters and cannot use the following characters : ‘ < > & \ “

Any information that is needed but not specified will be listed for selection, and before starting the job, you will be prompted to accept, reject, or modify the parameters of the job to be run, unless you have specified **yes**, in which case the job will be immediately sent to the scheduler.

On my system, when I enter a run command, I get the following prompt:

```
A job name must be specified.
The defined Job resources are:
  1: Matou
  2: Polymatou
  3: Rufus
  4: Minimatou
  5: Minou
  6: PmatouVerify
  7: MatouVerify
  8: RufusVerify
  9: Watchdog
Select Job resource (1-9):
```

If I then select number 5, I am prompted with:

```
Run Backup job
JobName: Minou
Fileset: Minou Full Set
Level: Incremental
Client: Minou
Storage: DLTDrive
Pool: Default
When: 2003-04-23 17:08:18
OK to run? (yes/mod/no):
```

If I now enter **yes**, the Job will be run. If I enter **mod**, I will be presented with the following prompt:

```
Parameters to modify:
  1: Level
  2: Storage
  3: Job
  4: Fileset
  5: Client
  6: When
```

(continues on next page)

(continued from previous page)

```
  7: Pool
Select parameter to modify (1-7):
```

If you wish to start a job at a later time, you can do so by setting the When time. Use the **mod** option and select **When** (no. 6). Then enter the desired start time in `YYYY-MM-DD HH:MM:SS` format.

The **spooldata** argument of the run command cannot be modified through the menu and is only accessible by setting its value on the intial command line. If no spooldata flag is set, the job, storage or schedule flag is used.

setbandwidth

This command is used to limit the bandwidth of a running job or a client.

```
setbandwidth limit=<nb> [ jobid=<id> | client=<cli> ]
```

The limit can be provided as a plain number, which would then mean the number of bytes per second, or a multiplier from among (case-insensitive) `k/s` (1,024), `kb/s` (1,000), `m/s` (1,048,576) or `mb/s` (1,000,000) can be appended.

This bandwidth control is applied to data transfers from File Daemon to Storage Daemon, and it may not result in as smooth a network bandwidth usage as traffic shaping at the network layer, but it requires no external facilities.

setdebug

This command is used to set the debug level in each daemon. The form of this command is:

```
setdebug level=nn [trace=0/1 client=<client-name> | dir | director |
storage=<storage-name> | all | options=0cidtTlp | tags=<tags>]
```

If **trace=1** is set, then tracing will be enabled, and the daemon will be placed in trace mode, which means that all debug output as set by the debug level will be directed to the file **bacula.trace** in the current directory of the daemon. Normally, tracing is needed only for Win32 clients where the debug output cannot be written to a terminal or redirected to a file. When tracing, each debug output message is appended to the trace file. You must explicitly delete the file when you are done.

If **options** parameter is set, the following arguments can be used to control debug functions.

0 clear debug flags

i Turn off, ignore bwrite() errors on restore on File Daemon

d Turn off decomp of BackupRead() streams on File Daemon

t Turn on timestamp in traces

T Turn off timestamp in traces

c Truncate trace file if trace file is activated

I Turn on recoding events on P() and V()

p Turn on the display of the event ring when doing a lockdump

It is now possible to use *class* of debug messages called `tags` to control the debug output of **Bacula** daemons.

all Display all debug messages

bvfs Display BVFS debug messages

sql Display SQL related debug messages

memory Display memory and poolmem allocation messages

scheduler Display scheduler related debug messages

```
* setdebug level=10 tags=bvfs,sql,memory
* setdebug level=10 tags=!bvfs
```

The `tags` option is composed of a list of tags, tags are separated by "," or "+" or "-" or "!". To disable a specific tag, use "-" or "!" in front of the tag.

setip

Sets new client address — if authorized.

A console is authorized to use the **SetIP** command only if it has a **Console** definition in both the Director and the Console. In addition, if the console name, provided on the **Name**, must be the same as a Client name, the user of that console is permitted to use the **SetIP** command to change the **Address** in the Director's *Client* resource to the IP address of the Console. This permits portables or other machines using DHCP (non-fixed IP addresses) to "notify" the Director of their current IP address.

show

The **show** command will list the records as defined in the Director's configuration file (normally **bacula-dir.conf**). This command is used mainly for debugging purposes by developers. The following keywords are accepted on the **show** command line: **catalogs**, **clients**, **counters**, **devices**, **directors filesets**, **jobs**, **messages**, **pools**, **schedules**, **storages**, **all**, **help**. Please don't confuse this command with the **list**, which displays the contents of the catalog.

sqlquery

The **sqlquery** command puts the Console program into SQL query mode where each line you enter is concatenated to the previous line until a semicolon (;) is seen. The semicolon terminates the command, which is then passed directly to the SQL database engine. When the output from the SQL engine is displayed, the formation of a new SQL command begins. To terminate SQL query mode and return to the Console command prompt, you enter a period (.) in column 1.

Using this command, you can query the SQL catalog database directly. Note you should really know what you are doing otherwise you could damage the catalog database. See the **query** command above for simpler and safer way of entering SQL queries.

Depending on what database engine you are using (*MySQL* or *PostgreSQL*), you will have somewhat different SQL commands available. For more detailed information, please refer to the *MySQL*, *PostgreSQL* documentation.

status

This command will display the status of all components. For the director, it will display the next jobs that are scheduled during the next 24 hours as well as the status of currently running jobs. For the Storage Daemon, you will have drive status or autochanger content. The File Daemon will give you information about current jobs like average speed or file accounting. The full form of this command is:

```
status [all | dir=<dir-name> | director [days=nnn] |
schedule [client=<client-name> | job=<job-name>] [days=nnn | limit=nnn]
client=<client-name> | [slots] storage=<storage-name>] | network [bytes=
↪<nb>]
```

If you do a **status dir**, the console will list any currently running jobs, a summary of all jobs scheduled to be run in the next 24 hours, and a listing of the last ten terminated jobs with their

statuses. The scheduled jobs summary will include the Volume name to be used. You should be aware of two things: 1. to obtain the volume name, the code goes through the same code that will be used when the job runs, but it does not do pruning nor recycling of Volumes; 2. The Volume listed is at best a guess. The Volume actually used may be different because of the time difference (more durations may expire when the job runs) and another job could completely fill the Volume requiring a new one.

In the Running Jobs listing, you may find the following types of information:

```
2507 Catalog MatouVerify.2004-03-13-05.05.02 is waiting execution
5349 Full    CatalogBackup.2004-03-13-01.10.00 is waiting for higher␣
↪priority jobs to finish
5348 Differe Minou.2004-03-13-01.05.09 is waiting on max Storage jobs
5343 Full    Rufus.2004-03-13-01.05.04 is running
```

Looking at the above listing from bottom to top, obviously JobId 5343 (Rufus) is running. JobId 5348 (Minou) is waiting for JobId 5343 to finish because it is using the *Storage* resource, hence the "waiting on max Storage jobs". JobId 5349 has a lower priority than all the other jobs so it is waiting for higher priority jobs to finish, and finally, JobId 2507 (MatouVerify) is waiting because only one job can run at a time, hence it is simply "waiting execution"

If you do a **status dir**, it will by default list the first occurrence of all jobs that are scheduled today and tomorrow. If you wish to see the jobs that are scheduled in the next three days (e.g. on Friday you want to see the first occurrence of what tapes are scheduled to be used on Friday, the weekend, and Monday), you can add the **days=3** option. Note, a **days=0** shows the first occurrence of jobs scheduled today only. If you have multiple run statements, the first occurrence of each run statement for the job will be displayed for the period specified.

If your job seems to be blocked, you can get a general idea of the problem by doing a **status dir**, but you can most often get a much more specific indication of the problem looking into the File and Storage Daemon status.

The **status schedule** provides more insight into the scheduler's world view. It will present a list of scheduled jobs, starting with the current day, and including past jobs of the current day.

By default, the Jobs for 10 days will be presented. Also, by default, the output will be limited to 100 items. Using the **days** and **limit** named options, which each take a numerical value, the output can be shortened. The **client** and **job** keywords take Client and Job names, respectively, and can be used to filter the Jobs to list. As of Bacula Enterprise 8.10 these keywords can appear multiple times, but that filters of the same type will be or'ed together, whereas Client and Job filters would both have to match. In other words, as each Job runs by definition on exactly one Client, combining client and job filters is pointless.

An example:

```
status schedule client=vanyar-fd client=GolgiApparat-fd limit=8

Scheduled Jobs (30/8):
Level          Type     Pri  Scheduled          Job Name          ␣
↪Schedule
================================================================================
Incremental    Backup    10  Tue 23-May 22:05   vanyar-psql       ␣
↪DailyPG
Incremental    Backup    10  Tue 23-May 23:05   GolgiApparat-all   Daily
Incremental    Backup    10  Tue 23-May 23:05   vanyar-all        ␣
↪DailyLC
```

(continues on next page)

(continued from previous page)

```
Incremental    Backup    10  Wed 24-May 22:05   vanyar-psql       ␣
↪DailyPG
Incremental    Backup    10  Wed 24-May 23:05   GolgiApparat-all   Daily
Incremental    Backup    10  Wed 24-May 23:05   vanyar-all        ␣
↪DailyLC
Incremental    Backup    10  Thu 25-May 22:05   vanyar-psql       ␣
↪DailyPG
Incremental    Backup    10  Thu 25-May 23:05   GolgiApparat-all   Daily
====
```

If you enter **status storage**, **Bacula** will prompt you with a list of the storage resources. When you select one, the Storage daemon will be requested to do a **status**. However, note that the Storage daemon will do a status of all the devices it has, and not just of the one you requested. In the current version of **Bacula**, when you enter the **status storage** command, it prompts you only with a subset of all the storage resources that the Director considers to be in different Storage daemons. In other words, it attempts to remove duplicate storage definitions. This can be a bit confusing at first, but can vastly simplify the promt listing if you have defined a large number of storage resources.

If you prefer to see the full list of all storage resources, simply add the keyword **select** to the command such as: **status select storage** and you will get a prompt that includes all storage resources even if they reference the same storage daemon.

If you enter **status network**, **Bacula** will prompt you with a list of the storage resources and a list of the client resources. Then, **Bacula** will test the network throughput between the two selected daemons.

```
*status network
The defined Client resources are:
  1: zog82-fd
  2: zog8-fd
Select Client (File daemon) resource (1-2): 2
Automatically selected Storage: File1
Connecting to Storage File1 at zog8:8103
Connecting to Client zog8-fd at zog8:8102
Running network test between Client=zog8-fd and Storage=File1 with 52.42␣
↪MB ...
2000 OK bytes=52428800 duration=48ms write-speed=1.088 GB/s 2000 OK␣
↪bytes=52428800 duration=56ms read-speed=933.8 MB/s
```

stop

The **stop** command is very similar to the **cancel** command with the main difference that the Job that is stopped is marked as Incomplete so that it can be restarted later by the *restart* command where it left off.

The JobId of the job to be stopped can be passed as a named parameter, such as `stop jobid=12345`.

The **stop** command with no arguments will prompt you with the list of running jobs allowing you to select one, which might look like the following:

```
*stop
Select Job:
    1: JobId=3 Job=Incremental.2012-03-26_12.04.26_07
    2: JobId=4 Job=Incremental.2012-03-26_12.04.30_08
```

(continues on next page)

(continued from previous page)

```
    3: JobId=5 Job=Incremental.2012-03-26_12.04.36_09
Choose Job to stop (1-3): 2
2001 Job "Incremental.2012-03-26_12.04.30_08" marked to be stopped.
3000 JobId=4 Job="Incremental.2012-03-26_12.04.30_08" marked to be stopped.
```

tag

The **tag** command will add, delete or list the tags associated with catalog records such as Clients, Jobs, Volumes or Objects. The command accepts all parameters in command line.

```
*tag add name="#test1" client=zog8-fd
1000 Tag added

*tag
Available Tag operations:
  1: Add
  2: Delete
  3: List
Select Tag operation (1-3): 1
Available Tag target:
  1: Client
  2: Job
  3: Volume
Select Tag target (1-3): 1
Automatically selected Client: zog8-fd
Enter the Tag value: test1
1000 Tag added
*tag
Available Tag operations:
  1: Add
  2: Delete
  3: List
Select Tag operation (1-3): 3
Available Tag target:
  1: Client
  2: Job
  3: Volume
Select Tag target (1-3): 1
Automatically selected Client: zog8-fd
+--------+----------+---------+
| tag    | clientid | client  |
+--------+----------+---------+
| test1  |        1 | zog8-fd |
| #test1 |        1 | zog8-fd |
+--------+----------+---------+
```

time

Prints the current time.

trace

Turn on/off trace to file.

umount

For old-time Unix guys. See the **unmount** command for full details.

unmount

This command causes the indicated **Bacula** Storage daemon to unmount the specified device. The forms of the command are the same as the **mount** command:

```
unmount storage=<storage-name> [ drive=<num> ]
```

```
unmount [ jobid=<id> | job=<job-name> ]
```

Once you unmount a storage device, **Bacula** will no longer be able to use it until you issue a mount command for that device. If **Bacula** needs to access that device, it will block and issue mount requests periodically to the operator.

If the device you are unmounting is an autochanger, it will unload the drive you have specified on the command line. If no drive is specified, it will assume drive 1.

update

This command will update the catalog for either a specific Pool record, a Volume record, or the Slots in an autochanger with barcode capability. In the case of updating a Pool record, the new information will be automatically taken from the corresponding Director's configuration resource record. It can be used to increase the maximum number of volumes permitted or to set a maximum number of volumes. The following main keywords may be specified: **media**, **volume**, **pool**, **slots**, **stats**, **jobid**.

In the case of updating a Job record, you can modify the **starttime**, the **client** and the **prune** attributes. **starttime** and **client** can be modified when doing a migration from an other backup software to Bacula for example. The **prune** attribute of the Job record is checked when trying to purge job or prune files from the catalog. If the **prune** Job's catalog attribute is 1 (Job record) or 2 (Files record), Bacula will not be able to purge the job record (or the file records) and recycle the associated volumes.

In the case of updating a Volume, you will be prompted for which value you wish to change. The following Volume parameters may be changed:

```
Volume Status
Volume Retention Period
Volume Use Duration
Maximum Volume Jobs
Maximum Volume Files
Maximum Volume Bytes
Recycle Flag
Recycle Pool
Slot
InChanger Flag
Pool
Volume Files
Volume from Pool
All Volumes from
Pool All Volumes from all Pools
```

For slots **update slots**, **Bacula** will obtain a list of slots and their barcodes from the Storage daemon, and for each barcode found, it will automatically update the slot in the catalog Media record to correspond to the new value. This is very useful if you have moved cassettes in the magazine, or if you have removed the magazine and inserted a different one. As the slot of each Volume is updated, the InChanger flag for that Volume will also be set, and any other Volumes in the Pool that were last mounted on the same Storage device will have their InChanger flag turned off. This permits **Bacula** to know what magazine (tape holder) is currently in the autochanger.

If you do not have barcodes, you can accomplish the same thing in version 1.33 and later by using the **update slots scan** command. The **scan** keyword tells **Bacula** to physically mount each tape and to read its VolumeName.

For Pool **update pool**, **Bacula** will move the Volume record from its existing pool to the pool specified.

For **Volume from Pool**, **All Volumes from Pool** and **All Volumes from all Pools**, the following values are updated from the Pool record: Recycle, RecyclePool, VolRetention, VolUseDuration, MaxVolJobs, MaxVolFiles, and MaxVolBytes. (RecyclePool feature is available with **Bacula** 2.1.4 or higher.)

The full form of the **update** command with all command line arguments is:

```
update volume=xxx pool=yyy slots volstatus=xxx VolRetention=ddd
VolUse=ddd MaxVolJobs=nnn MaxVolBytes=nnn Recycle=yes|no slot=nnn
enabled=n recyclepool=zzz actiononpurge=xxx
```

```
update volume=xxx frompool
```

```
update volume allfrompool=xxx
```

```
update volume fromallpools
```

use

This command allows you to specify which Catalog database to use. Normally, you will be using only one database so this will be done automatically. In the case that you are using more than one database, you can use this command to switch from one to another.

```
use [catalog=name-of-catalog]
```

var

This command takes a string or quoted string and does variable expansion on it the same way variable expansion is done on the **LabelFormat** string. Thus, for the most part, you can test your LabelFormat strings. The difference between the **var** command and the actual LabelFormat process is that during the var command, no job is running so "dummy" values are used in place of Job specific variables. Generally, however, you will get a good idea of what is going to happen in the real case.

version

The command prints the Director's version.

wait

The **wait** command causes the Director to pause until there are no jobs running. This command is useful in a batch situation such as regression testing where you wish to start a job and wait until that job completes before continuing. This command now has the following options:

```
wait [jobid=nn] [jobuid=unique id] [job=job name]
```

If specified with a specific JobId, the **wait** command will wait for that particular job to terminate before continuing.

Special dot Commands

There is a list of commands that are prefixed with a period (.). These commands are intended to be used either by batch programs or graphical user interface front-ends. They are not normally used by interactive users. Once GUI development begins, this list will be considerably expanded. More information about these commands can be found in src/dird/ua_dotcmds.c of the Bacula Community Project (www.bacula.org). The following is the list of dot commands:

```
.api
.backups job=xxx                    list backups for specified job
.clients                            list all client names
.catalogs                           list all catalog defined
.defaults client=xxx fileset=yyy    list defaults for specified client
.die                                cause the Director to segfault (for␣
↪debugging)
.dir                                when in tree mode prints the equivalent to␣
↪the dir
                                    command, but with fields separated by␣
↪commas rather
                                    than spaces.
.dump
.exit                               quit
.events                             list record custom events
.filesets                           list all fileset names
.help                               help command output
.jobs                               list all job names
.estimate                           estimate the size of the next job
.jlist                              list catalog objects in JSON format (see␣
↪list command)
.levels                             list all levels
.messages                           get quick messages
.msgs                               return any queued messages
.pools                              list all pool names
.quit                               quit
.putfile                            upload a PluginRestore object to the␣
↪director
.schedule                           list all schedule resources
.sql
.status                             get status output
.status dir header                  get header status output
.status dir running                 get running jobs status output
.status dir scheduled               get scheduled jobs status output
.status dir terminated              get terminated jobs status output
.storage                            return storage resource names
.volstatus                          list all possible volume statuses
.media                              list all media
.mediatypes                         list all defined mediatype
.locations
.actiononpurge                      list all possible values for ActionOnPurge␣
↪setting
.bvfs_lsdirs                        list directories in a directory for a␣
↪given set of jobs
.bvfs_lsfiles                       list files in a directory for a given set␣
```

(continues on next page)

(continued from previous page)

```
↪of jobs
.bvfs_get_volumes                  list volumes needed for a restore
.bvfs_update                       update the bvfs cache tables
.bvfs_get_jobids                   get jobids needed to restore a given job
.bvfs_get_jobs                     get job information
.bvfs_get_bootstrap                generate a bootstrap from a bvfs restore
.bvfs_get_fileindex                get the fileindex content for a given file
.bvfs_versions                     get all version of a file
.bvfs_get_delta                    get all delta parts of a file
.bvfs_restore                      generate a SQL table with all information␣
↪needed for a
                                   restore
.bvfs_cleanup                      cleanup the SQL restore table
.bvfs_decode_lstat                 decode the LSTAT field
.bvfs_clear_cache                  clear the BVFS cache of the catalog
.bvfs_update_fv                    compute extra statistics in the BVFS␣
↪tables (number of
                                   files and size)
.bvfs_delete_fileid                delete a given file
.setuid                            restrict BVFS queries with UID/GID
.ls                                list files on a client

.types                             list job types
.query
.tags                              list tags
```

Special At (@) Commands

Normally, all commands entered to the Console program are immediately forwarded to the Director, which may be on another machine, to be executed. However, there is a small list of **at** commands, all beginning with an at character (@), that will not be sent to the Director, but rather interpreted by the Console program directly. Note, these commands are implemented only in the tty console program and not in the *BAT* Console. These commands are:

- **@input <filename>** Read and execute the commands contained in the file specified.
- **@output <filename> w/a** Send all following output to the filename specified either overwriting the file (w) or appending to the file (a). To redirect the output to the terminal, simply enter **@output** without a filename specification. WARNING: be careful not to overwrite a valid file. A typical example during a regression test might be:

```
@output /dev/null
commands ...
@output
```

- **@tee <filename>w/a** Send all subsequent output to both the specified file and the terminal. It is turned off by specifying **@tall**, **@tee** or **@output** without a filename.
- **@tall <filename>w/a** Send all subsequent input and output to both the specified file and the terminal. It is turned off by specifying **@tall**, **@tee** or **@output** without a filename.
- **@sleep <seconds>** Sleep the specified number of seconds.
- **@time** Print the current time and date.

- **@version** Print the console's version.
- **@quit** quit
- **@exit** quit
- **@# anything** Comment
- **@help** Get the list of all special @ commands.
- **@separator <char>** When using `bconsole` with readline, you can set the command separator to one of those characters to write commands who require multiple input on one line, or to put multiple commands on a single line.

  ```
  !$%&'()*+,-/:;<>?[]^`{|}~
  ```

  Note, if you use a semicolon (;) as a separator character, which is common, you will not be able to use the **sql** command, which requires each command to be terminated by a semicolon.

Running the Console from a Shell Script

You can automate many Console tasks by running the console program from a shell script. For example, if you have created a file containing the following commands:

```
./bconsole -c ./bconsole.conf <<END_OF_DATA
unmount storage=DDS-4
quit
END_OF_DATA
```

when that file is executed, it will unmount the current DDS-4 storage device. You might want to run this command during a Job by using the `RunBeforeJob` or `RunAfterJob` records.

It is also possible to run the Console program from file input where the file contains the commands as follows:

```
./bconsole -c ./bconsole.conf <filename
```

where the file named `filename` contains any set of console commands.

As a real example, the following script is part of the **Bacula** regression tests. It labels a volume (a disk volume), runs a backup, then does a restore of the files saved.

```
bin/bconsole -c bin/bconsole.conf <<END_OF_DATA
@output /dev/null
messages
@output /tmp/log1.out
label volume=TestVolume001
run job=Client1 yes
wait
messages
@#
@# now do a restore
@#
@output /tmp/log2.out
restore current all
yes
```

(continues on next page)

(continued from previous page)

```
wait
messages
@output
quit
END_OF_DATA
```

The output from the backup is directed to `/tmp/log1.out` and the output from the restore is directed to `/tmp/log2.out`. To ensure that the backup and restore ran correctly, the output files are checked with:

```
grep "^ *Termination: *Backup OK" /tmp/log1.out
backupstat=$?
grep "^ *Termination: *Restore OK" /tmp/log2.out
restorestat=$?
```

Adding Volumes to a Pool

If you have used the **label** command to label a Volume, it will be automatically added to the Pool, and you will not need to add any media to the pool.

Alternatively, you may choose to add a number of Volumes to the pool without labeling them. At a later time when the Volume is requested by **Bacula** you will need to label it.

Before adding a volume, you must know the following information:

1. The name of the Pool (normally "Default")
2. The Media Type as specified in the `Storage` in the Director's configuration file (e.g. "DLT8000")
3. The number and names of the Volumes you wish to create.

For example, to add media to a Pool, you would issue the following commands to the console program:

```
*add
Enter name of Pool to add Volumes to: Default
Enter the Media Type: DLT8000
Enter number of Media volumes to create. Max=1000: 10
Enter base volume name: Save
Enter the starting number: 1
10 Volumes created in pool Default
*
```

To see what you have added, enter:

```
*list media pool=Default
+-------+----------+---------+---------+-------+------------------+
| MedId | VolumeNa | MediaTyp| VolStat | Bytes | LastWritten      |
+-------+----------+---------+---------+-------+------------------+
|    11 | Save0001 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    12 | Save0002 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    13 | Save0003 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    14 | Save0004 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    15 | Save0005 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    16 | Save0006 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    17 | Save0007 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
```

(continues on next page)

(continued from previous page)

```
|    18 | Save0008 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    19 | Save0009 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
|    20 | Save0010 | DLT8000 | Append  |     0 | 0000-00-00 00:00 |
+-------+----------+---------+---------+-------+------------------+
*
```

Notice that the console program automatically appended a number to the base Volume name that you specify (Save in this case). If you don't want it to append a number, you can simply answer 0 (zero) to the question "Enter number of Media volumes to create. Max=1000:", and in this case, it will create a single Volume with the exact name you specify.

