Bacula Enterprise Management

Bacula Systems Documentation

Contents

Contents

The following article aims at presenting various information on how to manage Bacula Enterprise.

As a central part of your system, Bacula services will report critical issues with network, disk space, or Catalog database in its Job logs.

It is important to do some system monitoring to allow for correct backup operations:

- monitor storage space
- monitor Job results
- monitor your infrastructure

You need to monitor space because Bacula cannot predict whether there will be enough room on tape or disks. You must ensure the available space is adequate for storing Jobs, but also on the Director as full filesystems will prohibit the backup from running correctly. For example, the spooling data is by default in `/opt/bacula/working`, the same for the spooled attributes. If this file system is full, Jobs will not succeed. You might want to use a tool like Nagios in order to create alarms if the pool space or partition space for `/opt` is low. Also remember that filesystems filled close to capacity are prone to have performance issues.

1 Daily Management

The following chapter provides details on jobs.

1.1 Job Levels, Types and Statuses

The following chapter presents descriptions on job levels, job types and job statuses.

Job Levels

The table below describes several levels for a job.

Level	Description
Backup levels	
F	Full backup: All files
I	Incremental: Files modified since last backup
D	Differential: Files modified since last full backup
S	Since: Not used
F	Virtual full backup
Verification levels	
C	Verify from Catalog
V	Verify: Init database
O	Verify volume to Catalog entries
d	Verify disk attributes to Catalog
A	Verify data on volume
Others	
B	Base level job
—	None: for Restore and Admin

Job Types

The table below describes several types for a job.

Type	Description
B	Backup Job
V	Verify Job
R	Restore Job
D	Admin job
C	Copy of a Job
c	Copy Job
M	A previous backup job that was migrated
g	Migration Job
A	Archive Job
S	Scan Job
U	Console program
I	Internal system “job”

Job Statuses

The table below describes several statuses for a job.

Status	Description
A	Job canceled by user
B	Job blocked
C	Job created but not yet running
D	Verify differences
E	Job terminated in error
F	Job waiting on File daemon
I	Incomplete Job
L	Committing data (last despool)
M	Job waiting for Mount
R	Job running
S	Job waiting on the Storage daemon
T	Job terminated normally
W	Job terminated normally with warnings
a	SD despooling attributes
c	Waiting for Client resource
d	Waiting for maximum jobs
e	Non-fatal error
f	Fatal error
i	Doing batch insert file records
j	Waiting for job resource
l	Doing data despooling
m	Waiting for new media
p	Waiting for higher priority jobs to finish
q	Queued waiting for device
s	Waiting for storage resource
t	Waiting for start time

I – Incomplete Jobs

An incomplete Job is a Backup Job which has been ended before completion in a situation where the Storage Daemon was able and allowed (per configuration) to commit data to Volumes and to the Catalog that allows the Job to be resumed later on. This usually happens when there is a communication problem between SD and FD, or when the File Daemon fails for some reason.

Note that backups done from dynamically created data sets, for example database dumps or ephemeral file system snapshots, may not be restartable despite the Storage Daemon considering the data to be sufficient for this purpose.

See the console commands restart and stop.

1.2 Tray Monitor

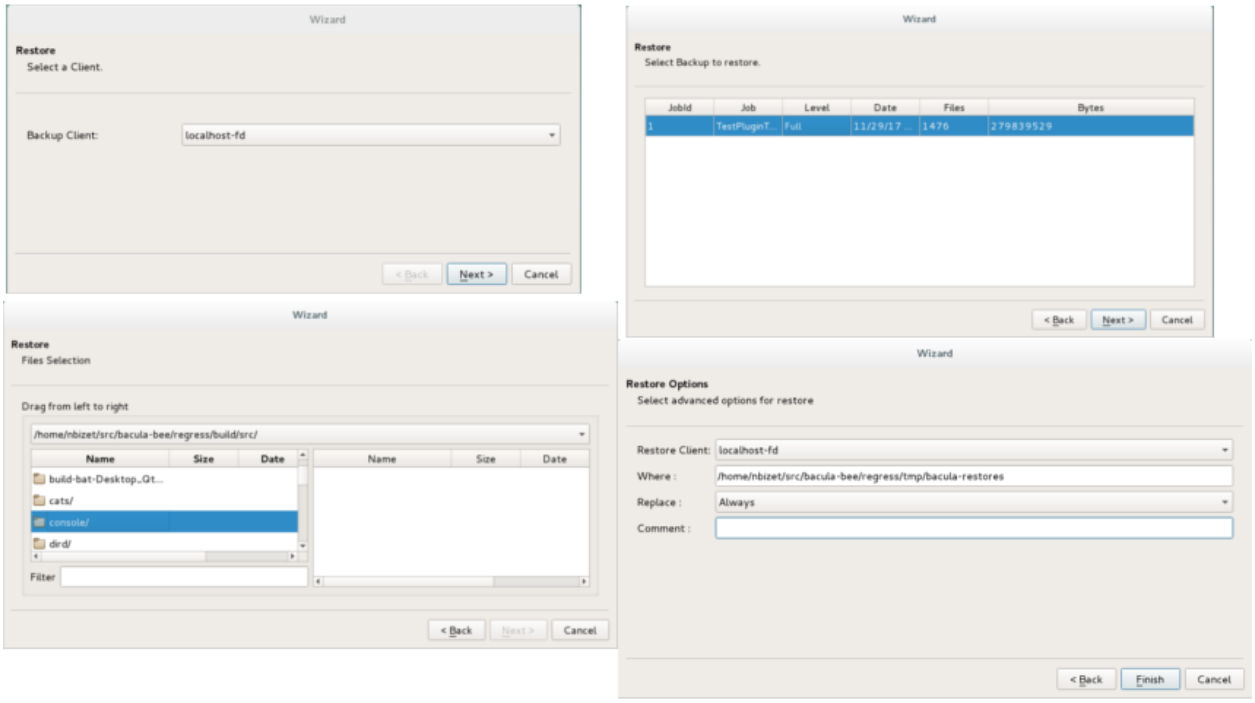
The Tray Monitor program is a simplified graphical user interface intended to be used with providing a tray area, similar to Windows' Task Bar information icon tray area.

It provides the capability to monitor all the core components of a **Bacula** infrastructure, namely Directors, Storage Daemons and File Daemons. The Tray Monitor program can be configured to monitor an arbitrary number of those daemons, and it can show an icon indicating current activity of the monitored daemon.

The Tray Monitor also has a built-in graphical configuration interface (but it can, as all **Bacula** components, be configured with a plain text file) and allows to initiate backup jobs through its context menu.

The Tray Monitor program can be used to interactively or non-interactively start jobs.

As of version 8.10 of Bacula Enterprise, the Tray Monitor can also initiate restore Jobs.



Non-interactively starting Jobs through the Tray Monitor program is particularly valuable if used in conjunction with the capability of File Daemons to serve as proxy when connecting to a Director (for Client Initiated Backups).

Installation

When building the **Bacula** software from source, the Tray Monitor program will be built similarly and under the same conditions as BAT. In particular, it needs a Qt build environment.

When installing the client software from packages, the Tray Monitor may be packaged individually or along with some other console programs – this is a packager's decision. Bacula Enterprise packages the Tray Monitor together with the `GUI` console on Linux, and with the client installer on Windows.

The actual steps to install depend on the platform and the packages used. A Windows Installer will typically allow to select the Tray Monitor for installation in some tree view of components to be installed.

The Tray Monitor program can be started whenever a `GUI` session is initiated, but care should be taken to use a configuration file that belongs to the user who runs the `GUI` session. In particular, having a default configuration file with credentials allowing unrestricted Director access usable by regular end users is to be avoided.

Configuration

The Tray Monitor program, unlike most other parts of the **Bacula** software, can be run without a configuration file in place. In this case, the configuration panel will be opened, allowing the user to configure the program according to her needs.

With the Tray Monitor it is quite likely that individual users will have their own configuration, i.e. a system-wide configuration file is not the common way to run the program. This is because the Tray Monitor, to do its work, needs access to backend components of the **Bacula** infrastructure, and individual users will only be provided with restricted access by the **Bacula** administration for security reasons.

For example, on a system shared by regular users and system administrators, the non-privileged users should not be able to see all jobs that can be run but only the ones relating to their own data, while a system administrator probably needs to monitor all the components running on the machines under his responsibility.

`GUI` Configuration

When started without a configuration file, or when the "Configure…" context menu item of the tray icon is clicked, the configuration window of the **Tray Monitor** program is shown. It consists of a tabbed view of configuration pages and some buttons, which provide the following functionality:

Save to save the current state into a text file (see paragraph above).

Cancel to abort the operation, leave everything as it was, and close the configuration panel.

Password to toggle between showing passwords on the current tab page in clear or hidden. This is useful to keep passwords hidden against observers, but still be able to check potentially complex and long passwords when needed.

(Add) Client to add a new, empty client configuration tab.

(Add) Storage to add a new, empty storage configuration tab.

(Add) Director to add a new, empty director configuration tab.

There will always be at least one tab to configure the identity of this Tray Monitor instance itself. Configuration page tabs that represent Clients, Storages, and Directors, will initially appear with a title of "New" which will change to the name of the monitored component once that has been provided.

The Monitor Configuration page

This tab defines the identity of the Tray Monitor instance. It is always available, cannot be removed, and there can only be one such tab. It contains the following fields:

Name the name of this Tray Monitor instance. This name is used to authenticate when connecting to any other **Bacula** daemon.

Note that, as usual with **Bacula**, the password used for authentication is configured individually for each counterpart.

The name is a *name-string* data type and allows only a limited set of characters (essentially, `ASCII` characters and numerals plus basic interpunction) and is limited to a length of 127 characters.

Refresh Interval is used to set the number of seconds between refreshes of status displays. Integer numbers between 5 and 9999 can be entered or selected.

Command Directory allows to enter or select the path of a directory which will be polled for command files. See below for details on how to use this feature.

Display Advanced Options will enable a settings page to select more Job options when the "Run…" context menu item of the **Tray Monitor** program is selected. See below for more information.

As usual for a **Bacula** component, unless the identity is configured the program will not be very functional.

Common elements

A common layout is used throughout all the configuration pages for the different **Bacula** Daemons. We will describe that first, before we look at specific additional configuration items for Client and Director.

We have always two groups of settings and a waste bin icon. The waste bin is used to delete the currently shown configuration from the Tray Monitor setup.

One group of settings is used to define the essential information to contact and use the component, and one group to configure `TLS`-related information. The former contains some required fields, among them the connection and authentication information, while the latter is optional.

We will not discuss the details of `TLS` configuration here – if TLS is required, the **Bacula** administrator will probably give detailed instructions and has to provide at least the "`CA` Certificate File". Simply put, leave the "Enabled" check box in the `TLS` section turned off unless detailed and site-specific configuration information is provided. If it is enabled, the selections in this group are critical and need to match what is configured for the respective daemon.

The General section contains the following elements common to Client, Storage, and Director configuration pages:

Name to configure the name that is used to represent this component in the Tray Monitor's user interface. This is a typical **Bacula** "name-string" with restrictions in length and usable character set (see above in the "Monitor Configuration" description).

Description can hold a textual description of the component. It is not used for any functionality, but rather to allow users to annotate their configuration.

The length of the description text is limited to 512 characters, and it can contain (nearly) any character as it's of the configuration data type "string".

Password is used to configure the shared secret that authenticates against the configured daemon. By default the text entered here is hidden, but it is possible to toggle with clear-text display using the "Password" button on the right.

The password is of **Bacula**'s "password" data type and as such can contain up to 127 characters. `ASCII` symbols, numerals, and some punctuation are allowed here.

Address is the address at which the configured daemon can be contacted. Usually, an IPv4 address in dotted quad notation or a DNS name is used.

Note that domain names should be fully qualified – using plain host names is strongly discouraged.

Support for IPv6 addresses depends on how the Tray Monitor program was built, but should be matching the IPv6 support status of all other **Bacula** programs resulting from the same build run.

Port represents the port number at which the daemon that is configured to be monitored can be contacted. Integer numbers between 1 and 65535 are allowed here. The default value shown in a new configuration page is the default port number for the daemon type.

This value should only be modified in two cases:

1. When the **Bacula** administrator explicitly indicates that a specific, non standard port number should be used.
2. When the File Daemon is configured for client-initiated backup. In this case the address to use will usually be "localhost", the port 9102, and the "Remote" checkbox will be ticked.

See below for more information on this mode of operation.

Timeout is used to set the connection timeout so the Tray Monitor will not try infinitely long to connect to the counterpart.

This is a typical **Bacula** "time-string", so the input format can be pretty flexible. However, for the purpose of network timeouts, plain integers representing numbers are suitable, and values between 5 and 300 (seconds) should be reasonable. The default unit is seconds.

Monitor is a checkbox which, when selected, tells the Tray Monitor program that the currently configured daemon should update the tray icon. When the daemon has some activity the tray icon will indicate this. Otherwise the tray icon will show an idle state.

This setting is most useful on client systems where it can give an indication of current activity to the desktop user, so that disrupting **Bacula** operations by shutting down or disconnecting from the network can be avoided.

Client configuration

In most cases, the **Bacula** File Daemon will be the one running on the local machine. In addition, a Client can also serve as a proxy for the Tray Monitor program to connect to a Director. This additional functionality is enabled with a configuration item:

Remote tells the Tray Monitor that this client can be used as a proxy to connect to a Director. This operation mode allows to initiate connections between Director and File Daemon from the client side, which is invaluable in situations where the DIR cannot (reliably) contact the File Daemon. Such a client-initiated connections can then be used to run jobs on clients inaccessible by the Director. See *Proxied Connection* for more details.

Director Configuration

In addition to the common configuration settings described above, one additional parameter is available here:

> **Use SetIp** causes the Tray Monitor program to send a `setip` command to the Director, which will enable the Director to communicate with this client.
>
> This setting can be useful in cases where the client computer has no fixed `IP` address or DNS name, but can be reached directly by the Director. Once the **setip** command was executed, the Director can use regular ways to operate on this client, for example starting scheduled Jobs or querying the Client status.
>
> For the **setip** command to succeed, it is required to use exactly identical Client and Console resource names in the Director's configuration. The Main manual describes this feature in detail.

Configuration file structure

As usual for **Bacula** components, the configuration for the Tray Monitor program is stored as a plain text file; the format is exactly the same as used with all the other components.

When you configure **Bacula** Tray Monitor with the `GUI`, a certain knowledge of the **Bacula** configuration scheme is surely helpful. When you edit the file with a text editor, this knowledge becomes essential.

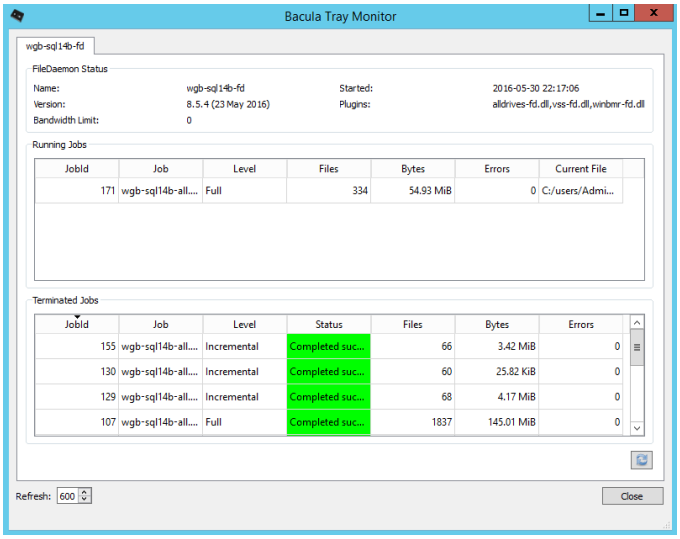
However, there is a rather simple mapping from fields in the `GUI` to configuration file contents: Each tab represents a resource of the specified type, and each actual setting is named identical or similarly in both the `GUI` and the file. For example, the following configuration file

```
Monitor {
    Name="wgb-sql14b-con"
    Refresh Interval = 1001
    Display Advanced Options = yes
}
Client {
    Name = "wgb-sql14b-fd"
    Address = "localhost"
    Password = "85test-onremote"
    Port = 9102
    Connect Timeout = 10
    Remote = yes
    Monitor = yes
}
```

exactly represents the settings in the `GUI` as seen in figure **blb:fig:tray-monitor-conf-mon-cli**.

Default locations of configuration files

The default locations of user specific configuration files that are written by the GUI are **/home/username/.bacula-tray-monitor.conf** (Linux) and **C:\Users\username\AppData\Roaming\bacula-tray-monitor.conf** (Windows, note that AppData is a hidden directory).



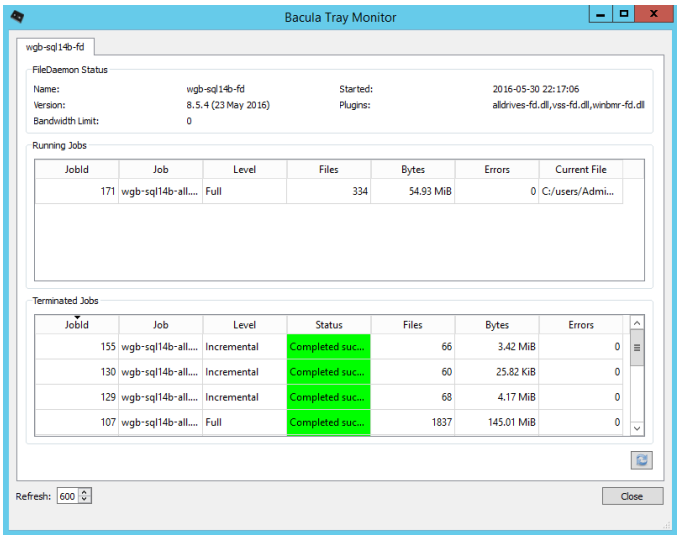
Monitoring

After double-clicking the Tray Monitor icon, or when selecting the "Display…" item in its context menu, the monitor screen is shown. In this screen, a tab selects which of the configured components of **Bacula** to show.

The monitoring panel consists of three areas, well known to any **Bacula** administrator, but probably not so easily understood by users without prior **Bacula** knowledge. On top, the general daemon status is shown. This gives information about the identity of the component, in particular the configured name, the version of the software, which plugins are currently loaded, and when the service process was started. In addition, some global settings may be shown – for a File Daemon, this is the configured Bandwidth Limit. Note that in-depth diagnostic information as presented by the **bconsole status** command is **not** shown.

Below the general status a list shows all currently running jobs. This list contains the JobID and the job's name, and as seen in figure, with the number of files and the amount of data already processed also gives some indication of job progress. The Error counter column, while looking important, is much less so than it appears, since actual error causes or messages can not be seen here.

Finally, the lowermost part of the monitoring panel shows an overview of the last jobs run by the particular daemon. Status text field is color coded (e.g. green for successful jobs).



Note that column headers can be clicked to change the sort order of the table, and column widths can be modified by dragging the table header column separators.

In the bottom right corner of the monitoring panel there is an icon to immediately refresh the display, while outside of the monitoring panel and the tab view, in the bottom left corner of the display window, the automatic refresh interval can be adjusted.

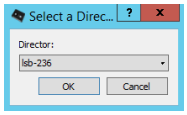
The "Close" button at the bottom right corner will close the display window in the same way as a click on the window decoration close button does.

Running Jobs

You can start a job by selecting "Run…" from the context menu of the tray icon.

If more than one Director or Clients with "Remote" capability are configured, a window will be shown to select the Director to use for the Job to be run. This window will appear as shown in *figure*. The name used in the selection drop-down menu is the one configured in the "Name:" setting of the component, see above in the "Configuration" section.

The "Run Job" window will be shown, which presents Job properties and an estimate of how much data will be backed up. The estimate will be updated when Job properties are changed. Be aware that this estimate is not based on current backup source information, i.e. it is not the result that you would get with the **estimate** command, but it is derived using a simple regression from Catalog information. The information presented is, however, usually enough to determine if the Job can finish during the remaining working hours.



Depending on the general Monitor configuration setting to "Display Advanced Options", a second, tabbed panel will be available and allow to change many other Job parameters, such as the backup level, the File Set to use, or the backup target like Pool and Storage device.

All settings are subject to access control restrictions configured at the Director, thus it is possible to have specific and secure options per each user (with each user connecting to their "own" access controlled Console resource).

Local Scheduling – the Command Directory

The Tray Monitor configuration item `Command Directory` can be used to automate **Bacula** activities through the Tray Monitor itself. In particular it allows to schedule jobs on the client machine, and not (only) inside the Director's configuration.

If configured with a path readable by the Tray Monitor program, the Tray Monitor, while running, will regularly look for files ending on and **".bcmd"** process them. Once a file was processed, it will be renamed to end with **.bcmd.ok** . These command files need to follow a specific syntax:

```
component-name: bconsole-command
```

`component-name` must be one of the defined Director or Client resources (Client only with **Remote=. Yes**).

`bconsole-command` is just a plain **bconsole** command which will be sent to the Director (possibly through the File Daemon).

The most useful command here is **run job=job-name** to initiate backups, where `job-name` would be the one that backs up from the local machine.

The solution to schedule backups of the local machine is accordingly to have or Windows' Task Scheduler create appropriate command files in the right location.

Part of that scheduled task could be to verify connectivity to the Director. An example suitable on a Linux or Unix system might look like this:

```
#!/bin/sh
if ping -c 1 director &> /dev/null
then
echo "my-dir: run job=backup" > /path/to/commands/backup.bcmd
fi
```

On a Windows system, a batch script like the following could be used:

```
@ECHO OFF
ping -n 1 -4 lsb-236.lsb.os.baculasystems.com >nul:
if ERRORLEVEL 1 (
echo Director not reachable!
) else (
echo wgb-sql14b-fd: run job=wgb-sql14b-all>%LOCALAPPDATA%\Bacula\commands\runnow.bcmd
)
```

Using **cron** to start this sort of script is pretty simple for a user with some Linux or Unix knowledge. Unfortunately, creating a scheduled task on a Windows machine is somewhat difficult to explain due to the number of required mouse clicks. We may provide step-by-step instructions at a later time.

Proxied Connection

With the Tray Monitor program providing a user friendly interface, and support for local scheduling, there's only one element missing to enable **Bacula** backing up computers that are unreliably available and may not be accessible from the Director itself: A way to enable the Director to run a Job on a given client, even if, from the Director, the client can not be accessed.

For many desktop and most laptop users, **Bacula** was traditionally not a very convenient solution, because it relied on the Director being configured with the client computer's IP address or host name, and then the Director being able to connect to the client.

In many corporate networks, however, there is neither DNS resolution nor static IP addressing in place for desktop computers. Furthermore, if the client computer is connected outside of the organization's LAN, for example in a home office behind a NAT'ing router, there will be no reliable way to connect to it at all.

In those situations, it is essential that the File Daemon connects to the Director, and the Director then uses this TCP connection to control the backup Job.

This is possible as of version 8.6 of using the *Client-Initiated Connection* feature. Note that both Director and File Daemon need to support this functionality. Also note that with newer versions, an alternative which is often easier to set up exists.

As we assume that such a setup of **Bacula** will usually be implemented to back up user computers, we will provide the documentation including a restricted console using .

We will first provide a description of the connection flow for a job that makes use of the proxied console connection, and then describe the required configuration in detail.

Connection Flow

For a Client-Initiated Backup to work through an FD-initiated connection, the following sequence of events will happen:

#. Console wants to start a Job, either manually or automated. The Director to run this job on should be the local FD, and, if the Tray Monitor is used, it should be configured with `Remote = Yes`.

#. The Console (most likely, the Tray Monitor program) connects to the local FD. Inside the FD configuration, such incoming connections are represented as a Director resource with two new settings: `Remote` and `Console`.

#. The Console submits the `proxy` command to the FD. Following this, the FD will use the Console resource referenced by the Director resource the current Console connection uses and use the provided information (Address, Name, and Password) to try to establish a connection to the "real" Director. If that succeeds, all further incoming console commands are directly passed on to that Director.

1. The Console submits the backup Job **run** command with the new option `fdcalled=1`.
2. The Director checks against configured inside its Console resource representing the current connection if the Job is allowed, and if it is, it first marks the existing connection from the FD as related to this Job, and then runs the Job.
3. When the DIR starts using the established connection for a Job, the FD closes the connection to the incoming console; the connection to the DIR is used exclusively for Job-related communication, and thus a console connected to it could not be used for anything useful from this moment on.
4. The Director, following its regular mode of operation, eventually sends a command to the FD telling it which Storage Daemon to contact for data transfers.
5. The File Daemon then connects to the SD as usual.
6. The Backup Job progresses as usual, and at the end, the FD will report the final Job status to the DIR.
7. Eventually, the `TCP` connection between FD and DIR will be properly shutdown.

The above step-by-step description already indicates where specific configuration for this operational mode is required:

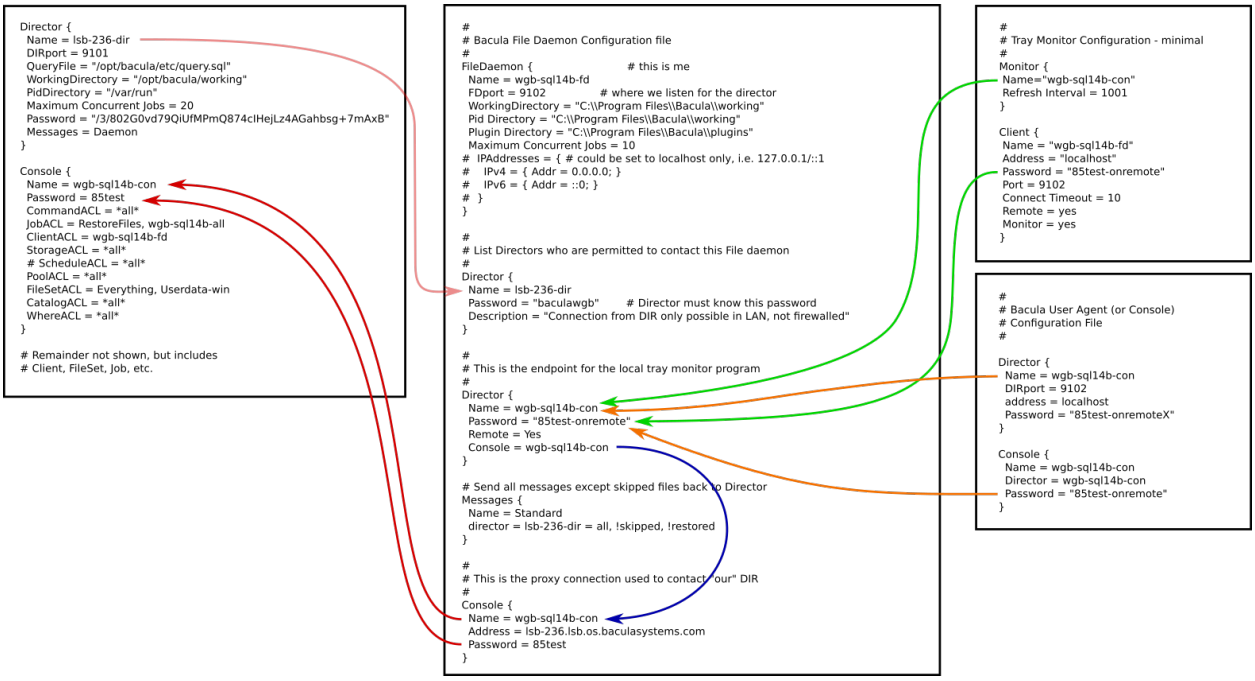
- The console program (either **bconsole** or the **Tray Monitor**) needs to be configured to connect to a File Daemon (usually the one running locally) instead of directly to a DIR.
- The File Daemon used needs to be configured with at least one additional Director resource with specific settings, one of them being the reference to a resource.
- In the Director's configuration, a client-specifc resource should be used, and this should be restricted in capabilities as far as possible.
- A Job that has to use a FD-initiated connection needs to be started with a specific new option, namely `fdcalled=1`.

For an experienced **Bacula** administrator, having to add and more resources to a File Daemon configuration will be unexpected. For new **Bacula** administrators, the complexity of the configuration scheme (which, as usual, is a result of the built-in flexibility) may be intimidating.

For this reason, we provide an overview of the required configuration in figure *SEE*.

In the overview figure, we have included a `bconsole` configuration which has not yet been mentioned. However, it works essentially identical to the Tray Monitor program connectivity, but allows – and requires! – the user to run arbitrary commands with all possible options. At this time, this is the only way to restore data using Client-Initiated connections.

The figure *SEE* shows all needed configuration entities, in some cases excerpts from the full configuration, in others the complete files.



Colored arrows indicate the relationships between resources. Most of them are also relationships between daemons potentially running on separate hosts, accordingly, any of those will require TCP/IP connectivity, including routing, name resolution, firewall and TCP Wrapper configuration. However, managing those aspects should not be an additional effort for **Bacula** administrators, as no additional requirements are introduced, nor will it be a problem for most users, as the common desktop configurations allow outgoing connections already.

Resource relationships

When looking at the configuration overview in *figure*, it is easy to spot distinct relationships. We will discuss them, starting at the console program used.

Console programs intended to use a proxied connection to the Director, or intended to start a Job through a client-initiated connection, need to be configured to contact the File Daemon used to serve as proxy to the Director, not the Director itself. Thus, the configuration will (usually) use **Address = localhost** and a port number of **9102**.

Note that **bconsole** is essentially agnostic of the fact that it connects to a File Daemon, not a Director, and thus uses a Director configuration resource, while the Tray Monitor program does know about those features and accordingly uses a slightly different configuration scheme with a Client resource with **Remote = Yes** set.

Also, **bconsole** and the Tray Monitor program use different resources to define themselves (which is where the name used for authentication is taken from), so there will be different "Identity" resources in use. It is possible and may be reasonable to use the same name and password, though, as the File Daemon can use the same resource to configure the properties of the connecting agent.

In *figure*, the relationship between Tray Monitor and (local) File Daemon configuration resources is shown in green, and the relationship from a **bconsole** configuration to that of the File Daemon is shown in orange.

File Daemon configuration for proxied connections to a Director are a bit more complex.

First, the File Daemon needs to accept incoming user agent connections from **bconsole** or the Tray Monitor program. However, these connections will (potentially) not only be used to query the FD's status, but also to submit more powerful commands, so a Monitor resource is not suitable.

Instead, the File Daemon has essentially to act as a Director for the user agent, so this functionality is configured in a resource.

This is essentially just a "normal" resource, but it is advisable to have distinct ones for "real" **Bacula** Directors as needed (usually there will only be one "real" Director) and for incoming user agents (also, there will usually be exactly one of them).

This is because direct Director connections may still be required in some cases, and it is surely advisable to have different passwords for a Director and for a user agent. Also, it is much easier to understand log files if different names are used for real Directors and user agent connections.

To enable the proxy functionality of the File Daemon, the directive **Remote** needs to be set to **yes** for those resources representing incoming user agent connections.

In addition, a directive should be present and indicate a resource used to contact the "real" Director – see below for details.

If no Console is referenced, the File Daemon will automatically select a Console it finds in its configuration, but since it is possible to have several of them defined, each with different properties, it is much safer and more clearly understandable what is configured if an explicit Console reference is used.

In the overview in figure *SEE*, this Console reference is indicated in blue.

The Console resource of the File Daemon specifies connection information and credentials to contact a "real" Director as usual.

It is advisable to use distinct names and passwords for each such Director connection, so that access can be controlled at least with the granularity of client machines.

However, if different users on a given machine will be using proxied connections to one or more Directors, there should be distinct Director resources per user, each using its own set of credentials.

As the File Daemon configuration file should be protected against user access, this allows separating user permissions and will allow to distinguish activity logged.

However, the local machine administrator will always be able to read all user credentials, thus needs to be trustworthy to at least the extent all the users combined are trusted.

Additional considerations apply to logging of File Daemon activity. Usually all File Daemon activity is logged to a default Director (Job-related activity is always logged to the initiating Director). However, in environments making use of the Client-Initiated Connection facilities, the central Director may not be a suitable target for the File Daemon messages as it will be inaccessible most of the time.

Thus we recommend to configure the File Daemon to write its messages to a local log file or logging daemon.

Director configuration for use in an environment using Client-Initiated Connections relies heavily on the use of Named Consoles with . In the configuration overview of *figure*, we show one such Console resource and red arrows from the File Daemon's resource indicate how it is referenced from the (proxying) File Daemon.

We also show the "regular" configuration, with the File Daemon being contacted by the Director (the Director-side Client resource is not in the overview) with a faint red arrow. The important part here is that authentication information is not shared between the two different connection direction setups.

Access Control Lists should be in place in the Director-side configuration for all Console resources that are used by non-administrative users. Only **Bacula** instance administrators should ever have full access to a Director.

Using the configuration scheme outlined, it is easily possible to have per-user Console configurations, typically restricting access to only the required resources. Most of the time, for example, there will be only one client and one backup Job allowed for such a Console connection.

In the overview graph we present a simplified, not very closely restricted Console resource.

In practice, much more restrictive configurations will be applied. We trust that a **Bacula** administrator will know which commands their users are supposed to execute, and which resources they need to access. However, with the Tray Monitor program, it may not be obvious which commands in particular are required, thus we provide the full list of mandatory commands here. Note that resource (for example, Job, Pool and Storage) restrictions need to be added.

A typical **Command ACL** directive for a Console used by a Tray Monitor which is allowed to run Jobs will require the following settings (without line breaks):

```
CommandACL = setbandwidth, proxy, run, restore, setip, wait,
   .status, .jobs, .clients .api, .storage, .pools, .filesets,
   .defaults, .estimate, .catalogs .bvfs_get_jobs,
   .bvfs_get_jobids, .bvfs_update, .bvfs_lsdirs, .bvfs_lsfiles
   .bvfs_restore, .bvfs_cleanup
```

Using bconsole for User-Initiated Jobs

Besides using the Tray Monitor utility to monitor and control a Director, it is also possible to use the console program **bconsole** with the File Daemon as a proxy, including the ability to start jobs which can use the File Daemon-initiated Director connection.

At the current time, this is actually the only reliable way to enable users with a restricted console connection and without full access from the Director to their desktop to restore from their backups.

To use these facilities, three items are important:

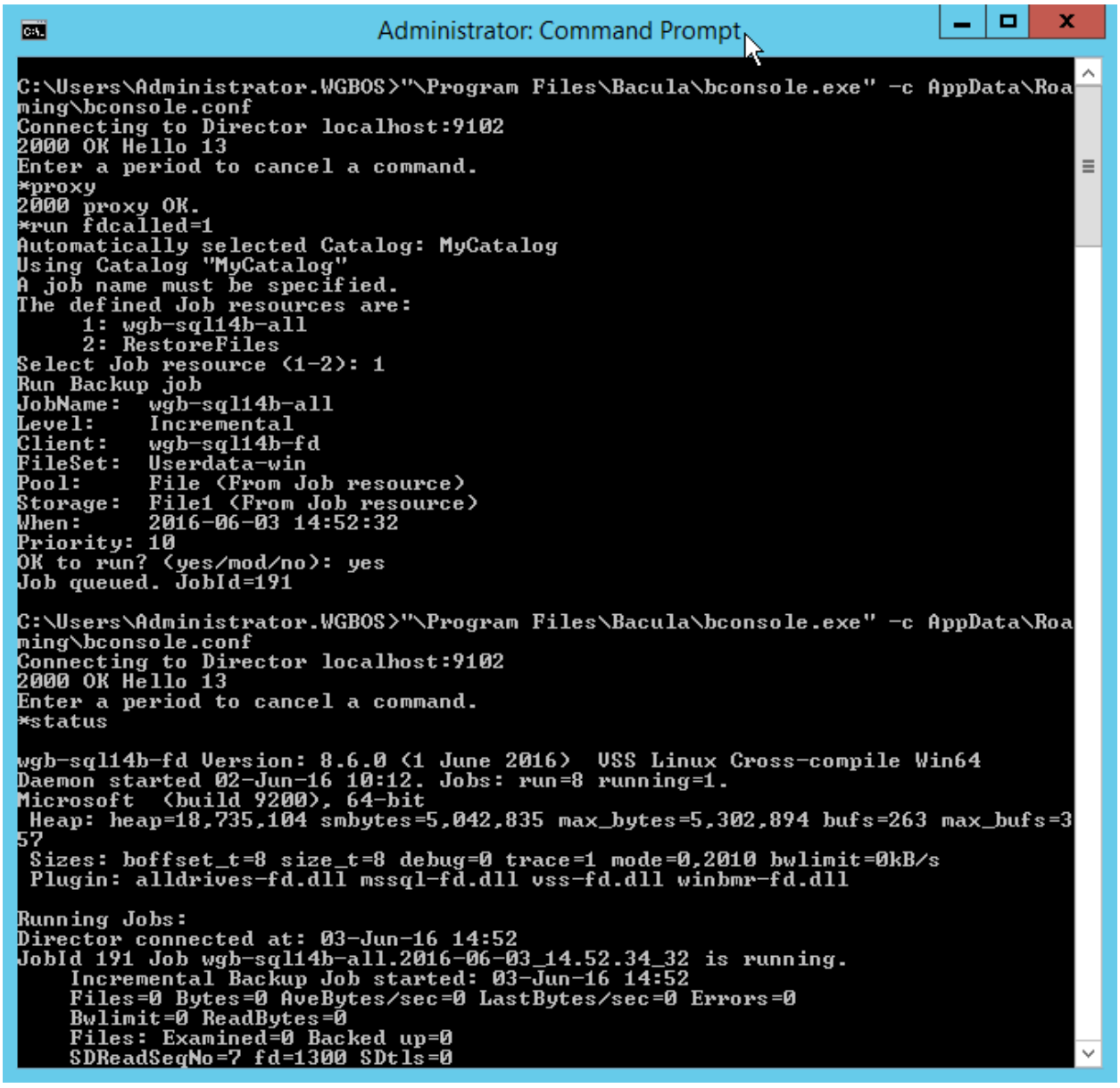
1. The programs involved need to be configured accordingly, i.e. as outlined above.
2. To initiate the proxied connection through the File Daemon, before any command intended for the Director is entered, the command `proxy` needs to be submitted. It should return with an "Ok" message.
3. Any job started that should use the established console connection to the Director needs the additional parameter `fdcalled=1` in its command line.

Note that the extra command and parameter are not required when submitting commands through the Tray Monitor program's command file facility. Also note that once a job has started, the console connection between File Daemon and **bconsole** will be terminated, causing **bconsole** to end. Job status can be observed with a new **bconsole** invocation; in fact, the status of the local File Daemon can then be observed without going through the Director. An example session is shown in figure *SEE*.

Supported Platforms

The Tray Monitor program is available for the following platforms:

- Microsoft Windows, both 32 and 64 bit, on all Windows versions under vendor support.
- Linux: Versions of Linux distributions supported by the Bacula Enterprise software with graphical desktops such as KDE, Gnome, or fvwm.



Command Line Options

The Tray Monitor program accepts the following command line options and parameters:

-c <filename> Configuration file to use. If the file does not exist, the program starts with an empty configuration and shows the configuration panel.

-d <number> Debug level. The higher the number, the more information is displayed.

-dt Print timestamps in debug output.

-t Test the configuration and exit. No output is a good sign.

-W <0|1> Force detection of systray capability. A zero indicates to run without tray icon, a one forces to use tray capability even if not detected.

-? Display short help and exit.

2 Logs Management

The following chapter provides information on how to manage logs.

2.1 Message Format Explanation

The messages generated by any Bacula component are all structured the same way:

- A message starts with an optional time stamp. The time stamp can be shown or hidden depending on command or options used. TODO details and links to list/llist, -td
- The name of the component generating the message follows. To be able to distinguish different message sources, convention is to use suffixes of *-fd*, *-sd*, *-dir* for File Daemon, Storage Daemon, and Director names, respectively.
- The next field of the message is the Job Id that caused the message. If messages are created that are not related to a Job, they get the Job Id 0 (zero) assigned. Such messages can refer to routine activity, such as volume pruning triggered by `status` commands, but they can also indicate noteworthy events, such as failing authentication due to an on-going brute force attack. Proper message resources will ensure that important messages are forwarded to responsible administrators and routine messages do not clutter logging facilities.
- The messages text follows. A messages Text can contain arbitrary characters (in particular, line breaks). Messages are intended for human readers, and a certain expertise with both Bacula and the systems involved is assumed.

  In recent versions of the Bacula software, a structured message characterisation has been introduced, which consists of a severity level keyword and a coded identifier at the beginning of a messages text. An example:

```
2022-11-14 18:18:10 bsys-demo-sd JobId 1984: Warning: [SW0201] Out of freespace␣
↪caused End of Volume "Vol-0045" at 245 on device "DiskAutochanger_Dev0" (/opt/
↪bacula/archive). Write of 64512 bytes got 3851.
```

  The text will remain complete and human readable, but the severity indication `Warning` and the message cause identifier `SW0201` make automated processing much easier.

  The identifier will always consist of two letters and four digits:

  - The first letter will be `D`, `S` or `F` for **Director**, **Storage Daemon** or **File Daemon**, respectively.
  - The second letter will indicate the severity level:

Character	Severity Level
A	Abort
F	Fatal
E	Error
W	Warning
S	Security
I	Informational
D	Debug
O	Ok, i.e. normal completion

Note that this list is a bit less granular than the message types documented in Messages Resource.

- The numerical identifier of the message will never change, even if the text will be adapted or translated.

  There is no particular structure in assignment of the numbers for each message, so filtering or processing by ranges is not suitable. Also, message id numbers will not be allocated sequentially.

Currently, not all messages have been converted to the new format, and this will only be completed after some more releases of the software.

3 Jobs Monitoring

Job monitoring is helpful when the Backup Administrator needs to decide what to do if a Job fails or if a warning occurs, as well as in sorting and prioritizing backup issues to resolve them in accordance with your environment. Joblogs and Bacula log file can be parsed by a monitoring software in order to re-run a job or take any action you may consider necessary (email/scripts/snapshots/reschedule the Job) if an error or a warning occurs.

3.1 Example Output

When you parse joblogs you can see:

```
Non-fatal FD errors:    0
SD Errors:              0
```

Non-fatal FD errors: 0 AND SD Errors: 0 Here everything went smoothly, no issues to report, probably no actions should be taken.

In the next case, you can see 1 Non-fatal FD errors and a “Backup OK – with warnings” termination:

```
Non-fatal FD errors:    1
SD Errors:              0
FD termination status:  OK
SD termination status:  OK
Termination:            Backup OK -- with warnings
```

Human investigation is advised for this job.

It should help to have a look at the joblog (`llist joblog jobid=<jobid>`). For the above Job, we might see the cause of the warnings, for example:


```
time: 2016-12-16 12:24:24
logtext: debianserver-fd JobId 18:      Could not stat "/home/bacula": ERR=No such file␣
↪or directory
```

Here we can see the `/home/bacula` directory specified in the job Fileset is not on the Client's filesystem to be backed up, as requested by the backup Job's Fileset.

Note: It is important to regularly check **all** of your joblogs to find potential problems and in case of error, update your error parsing scripts/software to take them into account and take appropriate action(s). A Job is correctly done when Backup Termination is "*Backup OK*" and Non-fatal FD/SD errors are equal to 0.

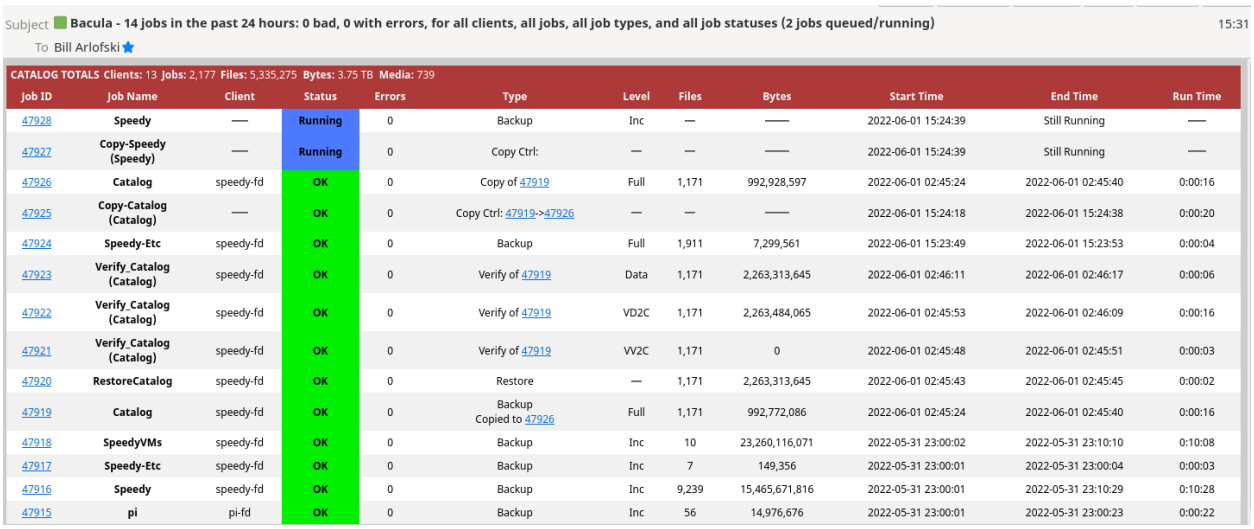
3.2 Third Party Scripts

Since Bacula started out as an open-source project, and this project and its surrounding community are still thriving, there are many third party scripts that can also help in the day to day monitoring of your Bacula Enterprise environment.

One such example of a third party script is called *baculabackupreport.py*.

This is a Python script that will send an HTML email report of the Jobs that have been run in the last 24 hours (default). When configured to run in a cron job, this script has many features to help you to have a clear, concise overview of your Bacula environment on a daily basis.

The latest version of this script may be found on Github here: https://github.com/waa



Subject Bacula - 14 jobs in the past 24 hours: 0 bad, 0 with errors, for all clients, all jobs, all job types, and all job statuses (2 jobs queued/running) 15:31

To Bill Arlofski

CATALOG TOTALS Clients: 13 Jobs: 2,177 Files: 5,335,275 Bytes: 3.75 TB Media: 739

Job ID	Job Name	Client	Status	Errors	Type	Level	Files	Bytes	Start Time	End Time	Run Time
47928	Speedy	—	Running	0	Backup	Inc	—	——	2022-06-01 15:24:39	Still Running	—
47927	Copy-Speedy (Speedy)	—	Running	0	Copy Ctrl:	—	—	——	2022-06-01 15:24:39	Still Running	—
47926	Catalog	speedy-fd	OK	0	Copy of 47919	Full	1,171	992,928,597	2022-06-01 02:45:24	2022-06-01 02:45:40	0:00:16
47925	Copy-Catalog (Catalog)	—	OK	0	Copy Ctrl: 47919->47926	—	—	——	2022-06-01 15:24:18	2022-06-01 15:24:38	0:00:20
47924	Speedy-Etc	speedy-fd	OK	0	Backup	Full	1,911	7,299,561	2022-06-01 15:23:49	2022-06-01 15:23:53	0:00:04
47923	Verify_Catalog (Catalog)	speedy-fd	OK	0	Verify of 47919	Data	1,171	2,263,313,645	2022-06-01 02:46:11	2022-06-01 02:46:17	0:00:06
47922	Verify_Catalog (Catalog)	speedy-fd	OK	0	Verify of 47919	VD2C	1,171	2,263,484,065	2022-06-01 02:45:53	2022-06-01 02:46:09	0:00:16
47921	Verify_Catalog (Catalog)	speedy-fd	OK	0	Verify of 47919	VV2C	1,171	0	2022-06-01 02:45:48	2022-06-01 02:45:51	0:00:03
47920	RestoreCatalog	speedy-fd	OK	0	Restore	—	1,171	2,263,313,645	2022-06-01 02:45:43	2022-06-01 02:45:45	0:00:02
47919	Catalog	speedy-fd	OK	0	Backup Copied to 47926	Full	1,171	992,772,086	2022-06-01 02:45:24	2022-06-01 02:45:40	0:00:16
47918	SpeedyVMs	speedy-fd	OK	0	Backup	Inc	10	23,260,116,071	2022-05-31 23:00:02	2022-05-31 23:10:10	0:10:08
47917	Speedy-Etc	speedy-fd	OK	0	Backup	Inc	7	149,356	2022-05-31 23:00:01	2022-05-31 23:00:04	0:00:03
47916	Speedy	speedy-fd	OK	0	Backup	Inc	9,239	15,465,671,816	2022-05-31 23:00:01	2022-05-31 23:10:29	0:10:28
47915	pi	pi-fd	OK	0	Backup	Inc	56	14,976,676	2022-05-31 23:00:01	2022-05-31 23:00:23	0:00:22

Fig. 1: Example of a report showing some backup jobs, some copy control jobs, and two running jobs. Notice that all JobIds are URL links to either BWeb Management Suite (Bacula Enterprise) or Baculum (Bacula Community).

3.3 Error Files Tracking

The FileEvent catalog table is currently filled with errors that occur during a Job, such as when I/O error hinders the successful backup of a file. The `list fileevents jobid=xx` bconsole command can be used to view the errors associated with a specific JobId.

See the Console commands.

3.4 Daemons Monitoring

In order to monitor daemons, use the Monitor configuration file. The Monitor configuration file is a stripped down version of the Director configuration file, mixed with a Console configuration file. It simply contains the information necessary to contact Directors, Clients, and Storage daemons you want to monitor.

For a general discussion of configuration file and resources including the data types recognized by **Bacula**, see the Configuration chapter.

The following Monitor Resource definition must be defined:

- *Monitor* – to define the Monitor's name used to connect to all the daemons and the password used to connect to the Directors. Note, you must not define more than one Monitor resource in the Monitor configuration file.
- At least one *Client*, *Storage* or *Director* resource, to define the daemons to monitor.

Monitor Resource

The Monitor resource defines the attributes of the Monitor running on the network. The parameters you define here must be configured as a Director resource in Clients and Storages configuration files, and as a Console resource in Directors configuration files.

Monitor Start of the Monitor records.

Name = <name> Specify the Director name used to connect to Client and Storage, and the Console name used to connect to Director. This record is required.

DisplayAdvancedOptions = <boolean> Display advanced options in the tray monitor (for backup and restore operations)

CommandDirectory = <directory> Directory where the tray monitor will look at a regular interval to find commands to execute.

Refresh Interval = <time> Specifies the time to wait between status requests to each daemon. It can't be set to less than 1 second, or more than 10 minutes, and the default value is **5 seconds**.

Director Resource

The Director resource defines the attributes of the Directors that are monitored by this Monitor.

As you are not permitted to define a Password in this resource, to avoid obtaining full Director privileges, you must create a Console resource in the Director's configuration file, using the Console Name and Password defined in the Monitor resource. To avoid security problems, you should configure this Console resource to allow access to no other daemons, and permit the use of only two commands: **status** and **.status** (see below for an example).

For more details, see BEConfigurationCustomizingConfFiles.

You may have multiple Director resource specifications in a single Monitor configuration file.

Director Start of the Director records.

Name = <name> The Director name used to identify the Director in the list of monitored daemons. It is not required to be the same as the one defined in the Director's configuration file. This record is required.

Port = <port-number> Specify the port to use to connect to the Director. This value will most likely already be set to the value you specified on the **–:with-baseport** option of the **./configure** command. This port must be identical to the **DIRport** specified in the Director** resource of the Director's configuration file. The default is **9101** so this record is not normally specified.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address used to connect to the Director. This record is required.

Client Resource

The Client resource defines the attributes of the Clients that are monitored by this Monitor.

You must create a Director resource in the Client's configuration file, using the Director Name defined in the Monitor resource. To avoid security problems, you should set the **Monitor** directive to **Yes** in this Director resource.

For more details, see BEConfigurationCustomizingConfFiles.

You may have multiple Director resource specifications in a single Monitor configuration file.

Client (or FileDaemon) Start of the Client records.

Name = <name> The Client name used to identify the Director in the list of monitored daemons. It is not required to be the same as the one defined in the Client's configuration file. This record is required.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address in dotted quad notation for a **Bacula** File daemon. This record is required.

Port = <port-number> Where the port is a port number at which the **Bacula** File daemon can be contacted. The default is **9102**.

Password = <password> This is the password to be used when establishing a connection with the File services, so the Client configuration file on the machine to be backed up must have the same password defined for this Director. This record is required.

Storage Resource

The Storage resource defines the attributes of the Storages that are monitored by this Monitor.

You must create a Director resource in the Storage's configuration file, using the Director Name defined in the Monitor resource. To avoid security problems, you should set the **Monitor** directive to **Yes** in this Director resource.

For more details, see BEConfigurationCustomizingConfFiles.

You may have multiple Director resource specifications in a single Monitor configuration file.

Storage Start of the Storage records.

Name = <name> The Storage name used to identify the Director in the list of monitored daemons. It is not required to be the same as the one defined in the Storage's configuration file. This record is required.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address in dotted quad notation for a **Bacula** Storage daemon. This record is required.

Port = <port> Where port is the port to use to contact the storage daemon for information and to start jobs. This same port number must appear in the Storage resource of the Storage daemon's configuration file. The default is **9103**.

Password = <password> This is the password to be used when establishing a connection with the Storage services. This same password also must appear in the Director resource of the Storage daemon's configuration file. This record is required.

Tray Monitor Security

There is no security problem in relaxing the permissions on **tray-monitor.conf** as long as FD, SD and DIR are configured properly, so the passwords contained in this file only gives access to the status of the daemons. It could be a security problem if you consider the status information as potentially dangerous.

Concerning Director's configuration: In **tray-monitor.conf**, the password in the Monitor resource must point to a restricted console in **bacula-dir.conf**.

For more details, see BEConfigurationCustomizingConfFiles.

So, if you use this password with bconsole, you'll only have access to the status of the director (commands **status** and **.status**). It could be a security problem if there is a bug in the ACL code of the director.

Concerning File and Storage Daemons' configuration: In **tray-monitor.conf**, the Name in the Monitor resource must point to a Director resource in **bacula-fd/sd.conf**, with the Monitor directive set to **yes** (once again, see the documentation). It could be a security problem if there is a bug in the code which check if a command is valid for a Monitor (this is very unlikely as the code is pretty simple).

Click below to see the Tray Monitor sample configuration.

Tray Monitor Sample Configuration

An example Tray Monitor configuration file might be the following:

```
#
# Bacula Tray Monitor Configuration File
#
Monitor {
    Name = rufus-mon # password for Directors
    RefreshInterval = 10 seconds
}
Client {
    Name = rufus-fd
    Address = rufus
    Port = 9102
    # password for FileDaemon
    Password = "FYpq4yyI1y562EMS35bA0J0QC0M2L3t5cZObxT3XQxgxppTn"
}
Storage {
    Name = rufus-sd
    Address = rufus
    Port = 9103
    # password for StorageDaemon
    Password = "9usxgc307dMbe7jbD16v0PXlhD64UVasIDD0DH2WAujcDsc6"
}
Director {
    Name = rufus-dir
    port = 9101
    address = rufus
}
```


Sample File Daemon's Director Record

Click here to see the full example.

```
#
# Restricted Director, used by tray-monitor to get the
#
status of the file daemon
#
Director {
    Name = rufus-mon
    Password = "FYpq4yyI1y562EMS35bA0J0QC0M2L3t5cZObxT3XQxgxppTn"
    Monitor = yes
}
```

Sample Storage Daemon's Director Record

Click here to see the full example.

```
#
# Restricted Director, used by tray-monitor to get the
#
status of the storage daemon
#
Director {
    Name = rufus-mon
    Password = "9usxgc307dMbe7jbD16v0PXlhD64UVasIDD0DH2WAujcDsc6"
    Monitor = yes
}
```

Sample Director's Console Record

Click here to see the full example.

```
#
# Restricted console used by tray-monitor to get the status of the director
#
Console {
    Name = Monitor
    Password = "GN0uRo7PTUmlMbqrJ2Gr1p0fk0HQJTxwnFyE4WSST3MWZseR"
    CommandACL = status, .status
}
```


3.5 Using Bacula Catalog to Grab Information

Bacula Catalog contains lots of information about your IT infrastructure, how many files, their size, the number of video or music files etc. Using **Bacula** catalog during the day to get them permit to save resources on your servers.

In this chapter, you will find tips and information to measure bacula efficiency and report statistics.

Job Statistics

If you want to have statistics on your backups to provide some SLA indicators, you could use a few `SQL` queries on the table to report how many:

- jobs have run
- jobs have been successful
- files have been backed up
- and so on.

However, these statistics are accurate only if your job retention is greater than your statistics period, i.e, if jobs are purged from the catalog, you won't be able to use them.

Since the introduction of the `jobhisto` catalog DB table, you can use the **update stats [days=num]** console command to fill this table with new Job records. If you want to be sure to take in account only **good jobs**, e.g. if one of your important job has failed but you have fixed the problem and restarted it on time, you probably want to delete the first *bad* job record and keep only the successful one. For that simply let your staff do the job, and update table after two or three days depending on your organization using the **[days=num]** option.

These statistics records aren't used for restoring, but mainly for capacity planning, billings, etc.

The **BWeb** interface provides a statistics module that can use this feature. You can also use third party tools (like Talend) or extract information by yourself.

The **Statistics Retention =** Director directive defines the length of time that **Bacula** will keep statistics job records in the Catalog database after the Job End time. When this time period expires, and if user runs **prune stats** command, **Bacula** will prune (remove) Job records that are older than the specified period.

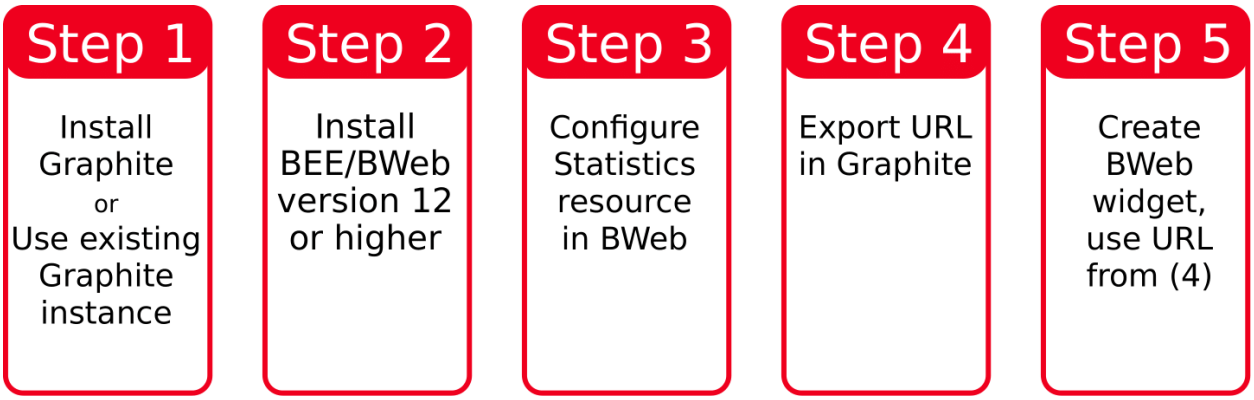
For example, you can use the following **RunScript** console commands in your nightly **BackupCatalog** job to maintain statistics.

```
Job {
    Name = BackupCatalog
    ...
    RunScript {
        Console = "update stats days=3"
        Console = "prune stats yes"
        RunsWhen = After
        RunsOnClient = no
        }
    }
```


3.6 BWeb with Graphite

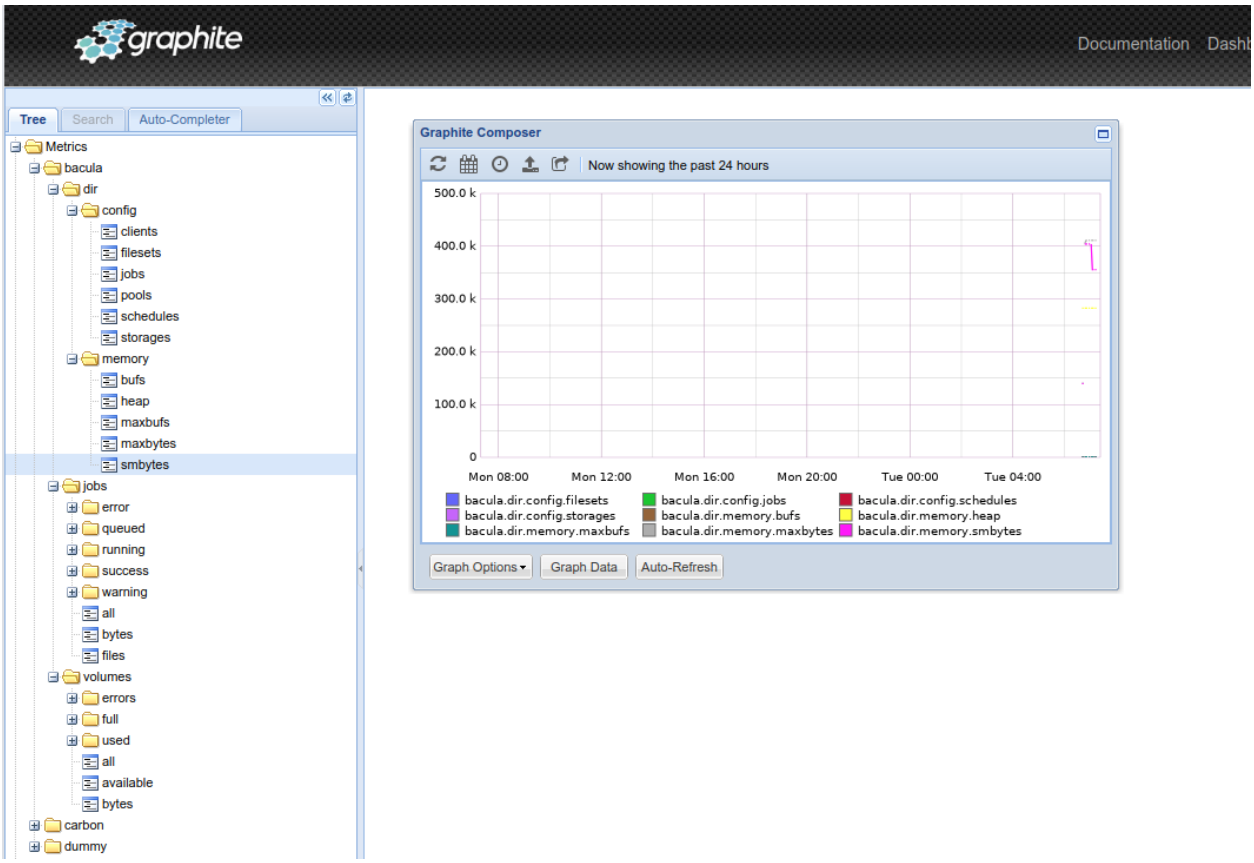
The following article explains how to customize your Bacula Enterprise installation with BWeb to use the powerful features of the Graphite visualization engine. Since version 12.0.0 BWeb can integrate Graphite widgets.

The workflow is as follows:



Graphite Installation

We will not explain in detail how to set up a Graphite server. The project's website https://graphiteapp.org/ has a Getting Started section where you can find further information.



Install Bacula Enterprise and BWeb Version 12.0.0 or Higher

Read the documents:

- BaculaEnterpriseInstallation
- BWeb Installation

and follow the instructions therein to install the Bacula Enterprise components.

Configure a Statistics Resource in BWeb

In the BWeb configuration overview screen (Workset) you can select one of the three major Bacula Components: Director, Storage Daemon, and File Daemon. One you have clicked on the button, the "Actions" box on the left contains a "List Statistics" button that will bring you to the Statistics resource overview page for the selected Bacula component.

Use the "Plus" symbol to add a new Statistics resource. It needs a Name (mandatory), optional description, you can set the interval with which Bacula will send information to the statsd process on your Graphite server. In the Type dropdown field select "Graphite" and enter metrics that you want to export e.g. bacula.jobs.*. A prefix can be useful to distinguish between your Bacula File Daemons for instance. Open the Graphite Options to enter the FQDN or IP address of your Graphite server and the port where Carbon receiver listens to plaintext.

Save the Statistics resource and restart the Bacula daemon that will then send statistics data to the Graphite host and port with the time interval specified.

It is possible to check if *statd* is running fine with bconsole:

```
.status dir statistics
```

In BWeb this can be done with the web console which you will have to activate in the top menu, Configuration, Configure BWeb, Edit, and then check "Web Bconsole". This will give you a new icon next to the search field in the top menu where you can access a bconsole to issue the above `.status` command.

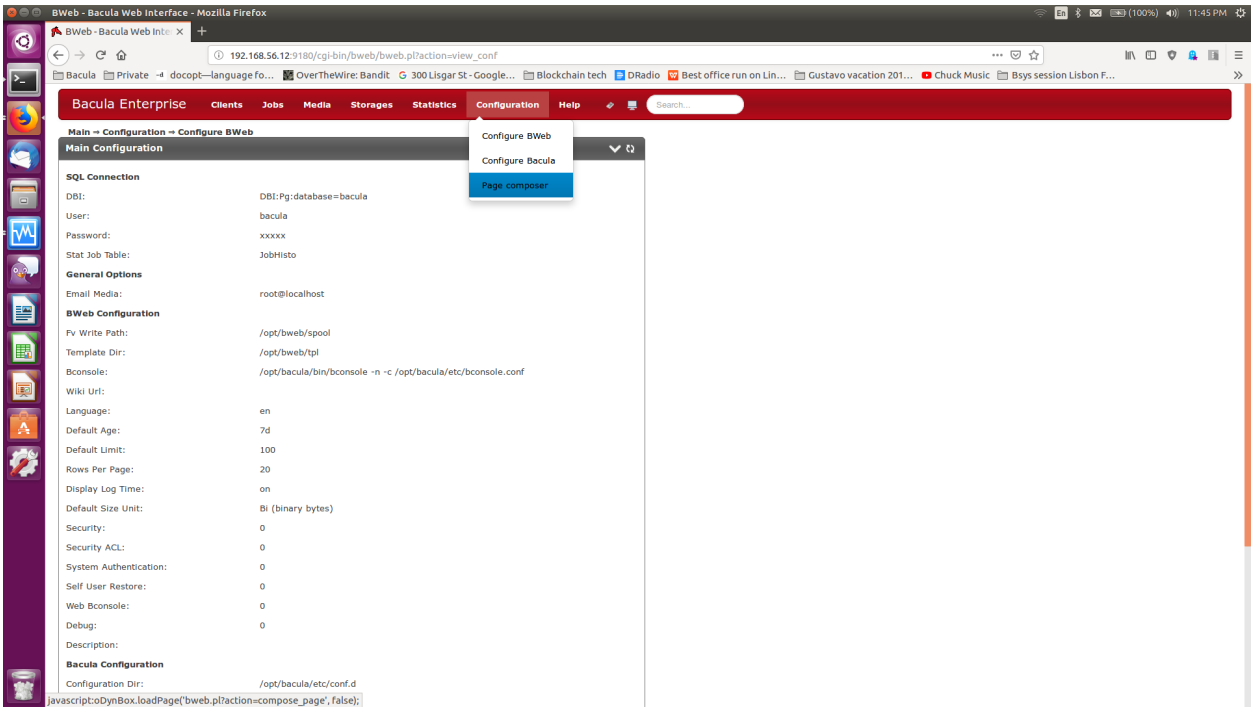
Browse Statistics Data in Graphite Web GUI and Export URL

In the Graphite web interface you will fine the new Bacula statistics in the Metrics section node tree structure on the left. Drill down to the metrics that you want to display and modify the graph to your liking. Once done, click the "Short URL" button in the Graphite Composer window to retrieve the URL to embed the statistics graph into the BWeb web interface.

Create Widget in BWeb and Add Graphite Metrics to Dashboard

In BWeb choose Configuration | Page Composer from the top menu. The button "Add new box" will allow you to create a custom graph based on the Graphite metrics: you choose a name for the new box/widget and an ID, then you enter the URL exported in step 4 and you can specify a custom refresh interval (each box in the BWeb dashboard can have its own refresh interval).

Confirm with "Add box" and drag and drop the new box into the page composer main window to modify your BWeb dashboard. You can change the position and size of the widgets.



3.7 Bacula Simple Network Management Protocol (bsnmp)

SNMP or Simple Network Management Protocol is a widely used protocol that helps to organize and collect information about managed devices on IP networks. SNMP exposes data from the managed devices in the form of variables that are organized in a management information base (MIB). This MIB is designed to describe status and configuration information. These variables can then be remotely accessed by centralized monitoring applications.

A given MIB offers a convention of variables that structure information in a way that is useful for the target system to be monitored. Different target systems that are used for the same purpose will share this structure, while reporting different values depending on their activities.

Bacula Enterprise activities can be remotely monitored through this protocol. Bacula Enterprise provides a simple script that can send SNMP notifications (SNMP INFORM) to a remote SNMP service. This service must be provided by any third-party application that offers this kind of monitoring service. The generated notifications can contain key data on how jobs are behaving, as job configuration data, job status and other values. This document describes how to setup this integration.

Note: This feature was added with Bacula version 16.0.

It is currently available as packaged add-on for a limited number of supported systems.

Instructions to install, test, and integrate the bsnmp job INFORM sender.

The functionality is provided by a very simple Python script which requires some MIBs to be available. Configuration is provided in a plain text file in ini format.

Script Installation

Note: pip must be installed prior to the installation of the bsnmp script.

1. Put the script file bsnmp.py into a location on the DIR host where the DIR can execute programs in. We propose to use /opt/bacula/scripts
2. Make the file executable by the Bacula DIR user account, usually `bacula`:

```
chown bacula. /opt/bacula/scripts/bsnmp.py
chmod u+x /opt/bacula/scripts/bsnmp.py
```

3. Ensure python3 is available: `python3 --version` should not fail.
4. Ensure the needed dependencies are installed. This can be somewhat difficult and may depend on distribution and python installation. In typical cases, with package based installations, but you may need to install the modules using pip:

```
# python3
Python 3.9.21 (main, Feb 10 2025, 00:00:00)
[GCC 11.5.0 20240719 (Red Hat 11.5.0-5)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import argparse
>>> import configparser
>>> import pyasn1.type.error
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'pyasn1'
>>> import asyncio
>>> import pysnmp
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'pysnmp'
```

The bsnmp script uses the SNMP library for Python (pysnmplib).

The Python modules required can be installed using pip:

```
# pip install pysnmplib
Collecting pysnmplib
  Downloading pysnmplib-5.0.24-py3-none-any.whl (286 kB)
     ---------------------------------- 286 kB 2.8 MB/s
Collecting pysnmp-pysmi<2.0.0,>=1.0.4
  Downloading pysnmp_pysmi-1.1.12-py3-none-any.whl (79 kB)
     ---------------------------------- 79 kB 632 kB/s
Collecting pycryptodomex<4.0.0,>=3.11.0
  Downloading pycryptodomex-3.23.0-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_
↪x86_64.whl (2.3 MB)
     ---------------------------------- 2.3 MB 41.3 MB/s
Collecting pysnmp-pyasn1<2.0.0,>=1.1.3
  Downloading pysnmp_pyasn1-1.1.3-py3-none-any.whl (77 kB)
     ---------------------------------- 77 kB 8.2 MB/s
Collecting requests<3.0.0,>=2.31.0
  Downloading requests-2.32.4-py3-none-any.whl (64 kB)
```

(continues on next page)

(continued from previous page)

```
     ---------------------------------- 64 kB 3.7 MB/s
Collecting ply<4.0,>=3.11
  Downloading ply-3.11-py2.py3-none-any.whl (49 kB)
     ---------------------------------- 49 kB 5.6 MB/s
Collecting certifi>=2017.4.17
  Downloading certifi-2025.6.15-py3-none-any.whl (157 kB)
     ---------------------------------- 157 kB 31.4 MB/s
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/lib/python3.9/site-
↪packages (from requests<3.0.0,>=2.31.0->pysnmp-pysmi<2.0.0,>=1.0.4->pysnmplib) (1.
↪26.5)
Collecting charset_normalizer<4,>=2
  Downloading charset_normalizer-3.4.2-cp39-cp39-manylinux_2_17_x86_64.
↪manylinux2014_x86_64.whl (149 kB)
     ---------------------------------- 149 kB 26.8 MB/s
Requirement already satisfied: idna<4,>=2.5 in /usr/lib/python3.9/site-packages␣
↪(from requests<3.0.0,>=2.31.0->pysnmp-pysmi<2.0.0,>=1.0.4->pysnmplib) (2.10)
Installing collected packages: charset-normalizer, certifi, requests, ply, pysnmp-
↪pysmi, pysnmp-pyasn1, pycryptodomex, pysnmplib
Successfully installed certifi-2025.6.15 charset-normalizer-3.4.2 ply-3.11␣
↪pycryptodomex-3.23.0 pysnmp-pyasn1-1.1.3 pysnmp-pysmi-1.1.12 pysnmplib-5.0.24␣
↪requests-2.32.4
```

Confirm if the libraries have been successfully installed:

```
# python3
Python 3.9.21 (main, Feb 10 2025, 00:00:00)
[GCC 11.5.0 20240719 (Red Hat 11.5.0-5)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import argparse
>>> import configparser
>>> import pyasn1.type.error
>>> import pysnmp
```

MIB Installation

1. MIBs for basic snmp infrastructure may be needed. Installation from distribution repos or tools will usually be sufficient. On a Debian (based) system, adding or enabling the non-free distribution repository and running `apt-get install snmp-mibs-downloader` will usually be sufficient.
2. The two provided MIB files should be copied to the system wide MIB repository, or to the pysnmp library directory for the MIBs:

```
# cp -iv ITS*MIB.py /usr/local/lib/python3.9/site-packages/pysnmp/smi/mibs
# chown root. /usr/local/lib/python3.9/site-packages/pysnmp/smi/mibs/ITS-*
# chmod 644 /usr/local/lib/python3.9/site-packages/pysnmp/smi/mibs/ITS-*
```


Script Configuration

1. The script needs a configuration file where it can be read by the Bacula DIR user. We suggest copying the provided example configuration to a suitable location:

```
# cp -a ITS-*.py /usr/local/lib/python3.9/site-packages/pysnmp/smi/mibs
# chown root. /usr/local/lib/python3.9/site-packages/pysnmp/smi/mibs/ITS-*
# chmod 644 /usr/local/lib/python3.9/site-packages/pysnmp/smi/mibs/ITS-*

  cp -iv bsnmp.conf /opt/bacula/etc/
  chown bacula. /opt/bacula/etc/bsnmp.conf
```

2. The configuration needs to be adapted. Using your favourite plain text editor, edit it:

```
[thisengine]
snmp_version = Valid values are 1, 2c or 3
username = USM user name on TRAP/INFORM receiver system
authkey = passphrase used to generate the SNPMv3 authkey
privkey = passphrase used to generate the SNPMv3 privkey

[target]
ipv4 host = DNS or IP address of trap receiver. IPv4 only
udp port = 162
```

   The information to put will, naturally, be very site specific and may need to be provided by a different team/person. Note that, at this time:

   - SNMPv1, SNMPv2c and SNMPv3 are supported. It is strongly recommended to use SNMPv3.
   - *only* the USM scheme is supported (we *may* support more in the future, but USM still seems to be the typical case)
   - *only* HMAC-SHA authentication and AES-CFB-128 encryption are supported. This will probably be extended in the future.
   - *only* UDP transport is supported. This should not be a problem. The port number is optional, default is 162, which is the assigned port number, and thus should not need modification in most cases.

Verification of Operations

Assuming the installation and configuration has been done per the above suggestions, running, as the `bacula` user,

```
/opt/bacula/scripts/bsnmp.py -c /opt/bacula/etc/bsnmp.conf -T
```

should result in no console output, and an SNMP INFORM message received at the configured receiver:

```
2025-06-18 10:01:19 bacula-dir.supportlab.lan [UDP: [10.0.98.59]:59092->[10.0.99.
↪40]:162]:
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (0) 0:00:00.00     SNMPv2-
↪MIB::snmpTrapOID.0 = OID: SNMPv2-SMI::enterprises.21267.8.1.0.1  SNMPv2-
↪SMI::enterprises.21267.8.1.1.1 = STRING: "Test-Job-Ignore"       SNMPv2-
↪SMI::enterprises.21267.8.1.1.2 = INTEGER: 4
```

Any console output should be reported so we can assist or fix what needs fixing.

Integration with DIR Messages Resource

1. Identify the configuration that defines the **Messages** for the Jobs that should trigger SNMP INFORMs to be sent.
2. Modify the **Messages** Resuource so that the `bsnmp.py` script call is added as a new **Mail** message destination. In a fairly default installation, this could be

```
Messages {
  Name = "Default"
  MailCommand = "/opt/bacula/bin/bsmtp -h localhost -f \"(Bacula) <%r>\" -s \
↪"Bacula: %t %e of %c %l\" %r"
  OperatorCommand = "/opt/bacula/bin/bsmtp -h localhost -f \"(Bacula) <%r>\" -s \
↪"Bacula: Intervention needed for %j\" %r"
  Mail  = "root@localhost" = All,!Skipped
  MailCommand = "/opt/bacula/scripts/bsnmp.py -c /opt/bacula/etc/bsnmp.conf"
  Mail = snmp-dummy = All,!Debug,!Saved,!Skipped
  Append  = "/opt/bacula/log/bacula.log" = All,!Skipped
  Console  = All,!Skipped
  Operator  = "root@localhost" = Mount
  Catalog  = All,Events
}
```

   The key points to observe are

   - The original Mail sending can be left as is, but all the entries for the new delivery method must follow the existing ones. So, `MailCommand` and `Mail` lines are added after the existing mail related entries.
   - The actual address used with the `bsnmp` script is ignored, but needs to be provided per syntax definition. Put whatever you like.
   - The actual message types used are not critical, but make sure that final Job reports get delivered. File lists, debug output, events and any other message types are, currently, ignored.

3. Verify configuration syntax as usual (`bacula-dir -t`)
4. Reload configuration. In bconsole, `reload` command.
5. Run a Job. After finishing, an SNMP INFORM should be delivered. When testing with `snmptrapd` (see above), information like

```
2025-06-18 11:31:28 bacula-dir.supportlab.lan [UDP: [10.0.98.116]:38413->[10.0.99.
↪40]:162]:
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (0) 0:00:00.00     SNMPv2-
↪MIB::snmpTrapOID.0 = OID: SNMPv2-SMI::enterprises.21267.8.1.0.1  SNMPv2-
↪SMI::enterprises.21267.8.1.1.1 = STRING: "LinuxHome"     SNMPv2-SMI::enterprises.
↪21267.8.1.1.2 = INTEGER: 0
```

   should be logged.

Example `snmptrapd` Configuration for Testing Purposes

1. The following suggestions are intended to be useful for reasonably skilled system administrators willing to read and understand a manual page as well as willing to spend a bit of time experimenting and learning. If more directed help is needed, ask the experts (TM).

   We assume a management system ingesting SNMP INFORMs is already in place. It would not be Bacula specific anyway.

2. Install all possibly needed MIBs on the designated snmp receiver host as outlined above.

3. Create an `snmptrapd` user. Manual pages are useful, **snmp** understanding is as well. In short: Put appropriate createuser lines into `/var/lib/snmp/snmptrapd.conf` and (re) start `snmptrapd`. The lines will be converted to local, engine-specific auth lines. `createuser informsubmit SHA informpass AES` is what was used for the default config file we provide.

4. For easier readability, starting `snmptrapd` with all MIBs loaded may be useful. For that purpose, editing the systemd unit file is probably the simplest approach. Thus:

   ```
   systemctl show snmptrapd.service | grep ExecStart
   ```

   to find the actual program call, and

   ```
   systemctl edit snmptrapd.service
   ```

   to add the needed `-m ALL` option:

   ```
   [Service]
   ExecStart=
   ExecStart=/usr/sbin/snmptrapd -Lsd -f -m ALL -p /run/snmptrapd.pid
   ```

5. If things work out, after the `bsnmp -T` call, information such as

   ```
   Jul 18 13:57:37 bsys-docdev snmptrapd[23559]: 2022-07-18 13:57:37 gnom.os.its-
   ↪lehmann.de [UDP: [192.168.0.55]:52424->[192.168.0.44]:162]:
   DISMAN-EXPRESSION-MIB::sysUpTimeInstance = Timeticks: (0) 0:00:00.00        SNMPv2-
   ↪MIB::snmpTrapOID.0 = OID: ITS-Bacula-MIB::baculaNotifyJob        ITS-Bacula-
   ↪MIB::baculaJobName = STRING: Test-Job-Ignore        ITS-Bacula-
   ↪MIB::baculaJobResult = INTEGER: stopped(4)
   ```

   should be logged.

3.8 Send Joblogs, Alerts, Mount Requests via Email (bsmtp)

bsmtp is a simple mail transport program that permits more flexibility than the standard mail programs typically found on Unix systems. It can even be used on Windows machines.

It is called:

```
Usage: bsmtp [-f from] [-h mailhost] [-s subject] [-c copy] [recipient ...]
       -c          set the Cc: field
       -dnn        set debug level to nn
       -f          set the From: field
       -h          use mailhost:port as the bsmtp server
       -l          limit the lines accepted to nn
```

(continues on next page)

(continued from previous page)

```
      -s          set the Subject: field
      -?          print this message.
```

If the -f option is not specified, bsmtp will use your userid. If the option -h is not specified bsmtp will use the value in the environment variable bsmtpSERVER or if there is none localhost. By default port 25 is used.

If a line count limit is set with the -l option, bsmtp will not send an email with a body text exceeding that number of lines. This is especially useful for large restore job reports where the list of files restored might produce very long mails your mail-server would refuse or crash. However, be aware that you will probably suppress the job report and any error messages unless you check the log file written by the Director (see the messages resource in this manual for details).

recipients is a space separated list of email recipients.

The body of the email message is read from standard input.

An example of the use of bsmtp would be to put the following statement in the Messages resource of your bacula-dir.conf file. Note, these commands should appear on a single line each.

```
mailcommand = "/home/bacula/bin/bsmtp -h mail.domain.com -f \"\(Bacula\) %r\"
               -s \"Bacula: %t %e of %c %l\" %r"
operatorcommand = "/home/bacula/bin/bsmtp -h mail.domain.com -f \"\(Bacula\) %r\"
                  -s \"Bacula: Intervention needed for %j\" %r"
```

Where you replace /home/bacula/bin with the path to your Bacula binary directory, and you replace mail.domain.com with the fully qualified name of your bsmtp (email) server, which normally listens on port 25. For more details on the substitution characters (e.g. %r) used in the above line, please see the documentation of the Mail Command in the Messages Resource chapter of the Bacula Enterprise Main manual.

It is highly recommended that you test one or two cases by hand to make sure that the mailhost that you specified is correct and that it will accept your email requests. Since bsmtp always uses a TCP connection rather than writing in the spool file, you may find that your from address is being rejected because it does not contain a valid domain, or because your message is caught in your spam filtering rules. Generally, you should specify a fully qualified domain name in the from field, and depending on whether your bsmtp gateway is Exim or Sendmail, you may need to modify the syntax of the from part of the message. Please test.

When running bsmtp by hand, you will need to terminate the message by entering a ctl-d in column 1 of the last line.

If you are getting incorrect dates (e.g. 1970) and you are running with a non-English language setting, you might try adding a LANG=”en_US” immediately before the bsmtp call.

In general, bsmtp attempts to cleanup email addresses that you specify in the from, copy, mailhost, and recipient fields, by adding the necessary < and > characters around the address part. However, if you include a display-name (see RFC 5332), some SMTP servers such as Exchange may not accept the message if the display-name is also included in < and >. As mentioned above, you must test, and if you run into this situation, you may manually add the < and > to the Bacula mailcommand or operatorcommand and when bsmtp is formatting an address if it already contains a < or > character, it will leave the address unchanged.

4 Storage Space Management

The following chapter provides information on how to manage storage space.

4.1 Volume Recycling

By default, once **Bacula** starts writing a Volume, it may append to the volume, but it will not overwrite the existing data thus destroying it. However when **Bacula recycles** a Volume, the Volume becomes available for being reused, and **Bacula** can at some later time overwrite the previous contents of that Volume. At that point all previous data on that Volume will be lost. If the Volume is a tape, the tape will be rewritten from the beginning. If the Volume is a disk file, the file will be truncated before being rewritten.

You may not want Bacula to automatically recycle (reuse) Volumes. Doing so may require a large number of Volumes though. However, it is also possible to manually recycle Volumes so that they may be reused. For more on manual recycling, see the section entitled *Manually Recycling Volumes*.

Most people prefer to have a Pool of Volumes that are used for daily backups and recycled once a week, another Pool of Volumes that are used for Full backups once a week and recycled monthly, and finally a Pool of Volumes that are used once a month and recycled after a year or two. With a scheme like this, the number of Volumes in your pool or pools remains constant.

By properly defining your Volume Pools with appropriate Retention periods, **Bacula** can manage the recycling (such as defined above) automatically.

Automatic Volume Recycling

Automatic recycling of Volumes is controlled by four records in the resource definition in the Director's configuration file. These four records are:

- AutoPrune = yes
- VolumeRetention = **<time>**
- Recycle = yes
- RecyclePool = **<APool>**
- ScratchPool = **<APool>**

The above first three directives are all you need assuming that you fill each of your Volumes then wait the Volume Retention period before reusing them, providing there is some non-pruned Jobs or Files on the Volume. **Recycle Pool** and **Scratch Pool** directives are not required. If not defined, **Bacula** will recycle Volumes and keep them in the current Pool.

If you want **Bacula** to stop using a Volume and recycle it before it is full, you will need to use one or more additional directives such as:

- **Use Volume Once=yes**
- **Volume Use Duration=ttt**
- **Maximum Volume Jobs=nnn**
- **Maximum Volume Bytes=mmm**

See below and the Basic Volume Management chapter for complete examples.

Automatic recycling of Volumes is performed by **Bacula** only when it wants a new Volume and no appendable Volumes are available in the Pool. It will then search the Pool for any Volumes with the flag set and the Volume Status is. At that point, it will choose the oldest purged volume and recycle it.

If there are no volumes with status, then the recycling occurs in two steps:

The first is that the Catalog for a Volume must be pruned of all Jobs (i.e. Purged) and Files contained on that Volume.

The second step is the actual recycling of the Volume. Only Volumes marked or will be considered for pruning. The Volume will be purged, all Jobs and Files associated to this Volume are pruned from Catalog, if the **VolumeRetention** period has expired. When a Volume is marked as , it means that no Catalog records for Jobs and Files reference that Volume, and the Volume can be recycled and reused. Please note a Volume can be reused even though the **VolumeRetention** period has not expired if the Jobs and Files associated to the Volume had been already pruned. Until recycling actually occurs, the Volume data remains intact. If no Volumes can be found for recycling for any of the reasons stated above, **Bacula** will request operator intervention (i.e. it will ask you to label a new volume).

A key point mentioned above, that can be a source of frustration, is that **Bacula** will only recycle purged Volumes if there is no other appendable Volume available, otherwise, it will always write to an appendable Volume before recycling even if there are Volumes marked as . This preserves your data as long as possible. So, if you wish to "force" **Bacula** to use a purged Volume, you must first ensure that no other Volume in the Pool is marked. If necessary, you can manually set a volume to. The reason for this is that **Bacula** wants to preserve the data on your old Volumes (even though purged from the catalog) as long as absolutely possible before overwriting it. There are also a number of directives such as **Volume Use Duration** that will automatically mark a volume as and thus no longer appendable.

Manually Recycling Volumes

Although automatic recycling of Volumes is implemented in version 1.20 and later (see the *Automatic Recycling of Volumes*), you may want to manually force reuse (recycling) of a Volume.

Assuming that you want to keep the Volume name, but you simply want to write new data on the tape, the steps to take are:

- Use the **update volume** command in the Console to ensure that the field is set to **1**
- Use the **purge jobs volume** command in the Console to mark the Volume as **Purged**. Check by using **list volumes**.

Once the Volume is marked Purged, it will be recycled the next time a Volume is needed.

If you wish to reuse the volume by giving it a new name, follow the following steps:

- Use the **purge jobs volume** command in the Console to mark the Volume as **Purged**. Check by using **list volumes**.
- Use the Console **relabel** command to relabel the Volume.

Please note that the **relabel** command applies only to tape Volumes.

For **Bacula** versions prior to 1.30 or to manually relabel the Volume, use the instructions below:

- Use the **delete volume** command in the Console to delete the Volume from the Catalog.
- If a different tape is mounted, use the **unmount** command, remove the tape, and insert the tape to be renamed.
- Write an EOF mark in the tape using the following commands:

```
mt -f /dev/nst0 rewind
mt -f /dev/nst0 weof
```

  where you replace **/dev/nst0** with the appropriate device name on your system.

- Use the **label** command to write a new label to the tape and to enter it in the catalog.

Be aware that the **delete** command can be dangerous. Once it is done, to recover the File records, you must either restore your database as it was before the **delete** command, or use the **bscan** utility program to scan the tape and recreate the database entries.

Specificities of Tape Recycling

Most people will want **Bacula** to fill a tape and when it is full, a new tape will be mounted, and so on. However, as an extreme example, it is possible for **Bacula** to write on a single tape, and every night to rewrite it. To get this to work, you must do two things: first, set the **VolumeRetention** to less than your save period (one day), and the second item is to make **Bacula** mark the tape as full after using it once. This is done using. If this latter record is not used and the tape is not full after the first time it is written, **Bacula** will simply append to the tape and eventually request another volume. Using the tape only once, forces the tape to be marked after each use, and the next time **Bacula** runs, it will recycle the tape.

An example Pool resource that does this is:

```
Pool {
    Name = DDS-4
    Use Volume Once = yes
    Pool Type = Backup
    AutoPrune = yes
    VolumeRetention = 12h # expire after 12 hours
    Recycle = yes
}
```

Regular Schedule Tape Usage Example

This example is meant to show you how one could define a fixed set of volumes that **Bacula** will rotate through on a regular schedule. There are an infinite number of such schemes, all of which have various advantages and disadvantages. Note: these volumes may either be tape volumes or disk volumes.

We start with the following assumptions:

- A single tape has more than enough capacity to do a full save.
- There are ten tapes that are used on a daily basis for incremental backups. They are prelabeled Daily1 ... Daily10.
- There are four tapes that are used on a weekly basis for full backups. They are labeled Week1 ... Week4.
- There are 12 tapes that are used on a monthly basis for full backups. They are numbered Month1 ... Month12
- A full backup is done every Saturday evening (tape inserted Friday evening before leaving work).
- No backups are done over the weekend (this is easy to change).
- The first Friday of each month, a Monthly tape is used for the Full backup.
- Incremental backups are done Monday - Friday (actually Tue-Fri mornings).

We start the system by doing a Full save to one of the weekly volumes or one of the monthly volumes. The next morning, we remove the tape and insert a Daily tape. Friday evening, we remove the Daily tape and insert the next tape in the Weekly series. Monday, we remove the Weekly tape and re-insert the Daily tape. On the first Friday of the next month, we insert the next Monthly tape in the series rather than a Weekly tape, then continue. When a Daily tape finally fills up, **Bacula** will request the next one in the series, and the next day when you notice the email message, you will mount it and **Bacula** will finish the unfinished incremental backup.

What does this give? Well, at any point, you will have the last complete Full save plus several Incremental saves. For any given file you want to recover (or your whole system), you will have a copy of that file every day for at least the last 14 days. For older versions, you will have at least three and probably four Friday full saves of that file, and going back further, you will have a copy of that file made on the beginning of the month for at least a year.

So you have copies of any file (or your whole system) for at least a year, but as you go back in time, the time between copies increases from daily to weekly to monthly.

What would the **Bacula** configuration look like to implement such a scheme?

```
Schedule {
    Name = "NightlySave"
    Run = Level=Full Pool=Monthly 1st sat at 03:05
    Run = Level=Full Pool=Weekly 2nd-5th sat at 03:05
    Run = Level=Incremental Pool=Daily tue-fri at 03:05
}
Job {
    Name = "NightlySave"
    Type = Backup
    Level = Full
    Client = LocalMachine
    Fileset = "File Set"
    Messages = Standard
    Storage = DDS-4
    Pool = Daily
    Schedule = "NightlySave"
}
# Definition of file storage device
Storage {
    Name = DDS-4
    Address = localhost
    SDPort = 9103
    Password = XXXXXXXXXXXXX
    Device = FileStorage
    Media Type = 8mm
}
Fileset {
    Name = "File Set"
    Include {
        Options { signature=MD5 }
        File = ffffffffffffffff
    }
    Exclude { File=*.o }
}
Pool {
    Name = Daily
    Pool Type = Backup
    AutoPrune = yes
    VolumeRetention = 10d    # recycle in 10 days
    Maximum Volumes = 10
    Recycle = yes
}
Pool {
    Name = Weekly
    Use Volume Once = yes
    Pool Type = Backup
    AutoPrune = yes
    VolumeRetention = 30d # recycle in 30 days (default)
    Recycle = yes
}
Pool {
```

(continues on next page)

(continued from previous page)

```
    Name = Monthly
    Use Volume Once = yes
    Pool Type = Backup
    AutoPrune = yes
    VolumeRetention = 365d # recycle in 1 year
    Recycle = yes
}
```

Recycling Algorithm

When a Job needs a Volume and no appendable, recycled or purged is available, **Bacula** starts the recycling algorithm pruning Jobs and Files for the oldest Volume in the pool used by the job. After the Volume is pruned, and if the **Recycle** flag is on (**Recycle=yes**) for that Volume, **Bacula** will relabel it and write new data on it.

As mentioned above, there are two key points for getting a Volume to be recycled. First, the Volume must no longer be marked **Append** (there are a number of directives to automatically make this change), and second since the last write on the Volume, one or more of the Retention periods must have expired so that there are no more catalog backup job records that reference that Volume. Once both those conditions are satisfied, the volume can be marked Purged and hence recycled.

The algorithm described below assumes that **AutoPrune** is enabled, that **Recycling** is turned on, and that you have defined appropriate Retention periods, or used the defaults for all these items.

- If the request is for an Autochanger device, look only for Volumes in the Autochanger (i.e. with **InChanger** set and that have the correct Storage device).
- Search the Pool for a Volume with *VolStatus=Append* (if there is more than one, the Volume with the oldest date last written is chosen. If two have the same date then the one with the lowest **MediaId** is chosen).
- Search the Pool for a Volume with *VolStatus=Recycle* and the **InChanger** flag is set true (if there is more than one, the Volume with the oldest date last written is chosen. If two have the same date then the one with the lowest **MediaId** is chosen).
- Search the Pool for a Volume with *VolStatus=Purged* and try recycling any purged Volumes.
- Prune volumes applying Volume retention period (Volumes with **VolStatus** or are pruned). Note, when all the File and Job records are pruned from a Volume, the Volume will be marked and this may be prior to the expiration of the Volume retention period.
- Prune the oldest Volume if **RecycleOldestVolume=yes** (the Volume with the oldest **LastWritten** date and **VolStatus** equal to or is chosen). This record ensures that all retention periods are properly respected.
- Purge the oldest Volume if **PurgeOldestVolume=yes** (the Volume with the oldest **LastWritten** date and **VolStatus** equal to or is chosen). We strongly recommend against the use of **PurgeOldestVolume** as it can quite easily lead to loss of current backup data.
- Search for a Volume in the Scratch Pool (the default Scratch pool or another Scratch Pool defined for the pool) and if found move it to the current Pool for the Job and use it. Note, when the Scratch Volume is moved into the current Pool, the basic Pool defaults are applied as if it is a newly labeled Volume (equivalent to an **update volume from pool** command).
- If we were looking for Volumes in the Autochanger, go back to step 2 above, but this time, look for any Volume whether or not it is in the Autochanger.
- Attempt to create a new Volume if automatic labeling is enabled and the maximum number of Volumes is not reached.
- Give up and ask operator.

The above occurs when **Bacula** has finished writing a Volume or when no Volume is present in the drive.

On the other hand, if you have inserted a different Volume after the last job, and **Bacula** recognizes the Volume as valid, the Storage daemon will request authorization from the Director to use this Volume. In this case, if you have set **Recycle CurrentVolume=yes** and the Volume is marked as Used or Full, **Bacula** will prune the volume and if all jobs were removed during the pruning (respecting the retention periods), the Volume will be recycled and re-used.

If you want to strictly observe the **Volume Retention Period** under all circumstances – i.e. you have a Volume with critical data, you must set **Recycle=no**. However, in doing so, you must manually do the recycling of the Volume for it to be used again.

The recycling algorithm in this case is:

- If the **VolStatus** is *Append* or **Recycle** is set, the volume will be used.
- If **Recycle Current Volume** is set and the volume is marked, or **Bacula** will prune the volume (applying the retention period). If all Jobs are pruned from the volume, it will be recycled.

This permits users to manually change the Volume every day and load volumes in an order different from what is in the catalog, and if the volume does not contain a current copy of your backup data, it will be used.

A few points to keep in mind:

1. If a pool doesn't have maximum volumes defined then **Bacula** will prefer to demand new volumes over forcibly purging older volumes.
2. If volumes become free through pruning, then they get marked as and are immediately available for recycling – these will be used in preference to creating new volumes.
3. If the Job, File, and Volume retention periods are different, then it's common to see a volume with no files or jobs listed in the database, but which is still not marked as **Purged**, if **Autoprune** is set to **no** in the pool resource.

Recycle Status

Each Volume inherits the Recycle status (yes or no) from the Pool resource record when the Media record is created (normally when the Volume is labeled). This Recycle status is stored in the Media record of the Catalog. Using the Console program, you may subsequently change the Recycle status for each Volume. For example in the following output from **list volumes**:

VolumeNa	Media	VolSta	VolByte	LastWritte	VolRet	Rec
File0001	File	Full	4190055	2002-05-25	14400	1
File0002	File	Full	1896460	2002-05-26	14400	1
File0003	File	Full	1896460	2002-05-26	14400	1
File0004	File	Full	1896460	2002-05-26	14400	1
File0005	File	Full	1896460	2002-05-26	14400	1
File0006	File	Full	1896460	2002-05-26	14400	1
File0007	File	Purged	1896466	2002-05-26	14400	1

all the volumes are marked as recyclable, and the last Volume, **File0007** has been purged, so it may be immediately recycled. The other volumes are all marked recyclable and when their Volume Retention period (14400 seconds or four hours) expires, they will be eligible for pruning, and possibly recycling. Even though Volume **File0007** has been purged, all the data on the Volume is still recoverable. A purged Volume simply means that there are no entries in the Catalog. Even if the Volume Status is changed to , the data on the Volume will be recoverable. The data is lost only when the Volume is re-labeled and re-written.

To modify Volume **File0001** so that it cannot be recycled, you use the **update volume pool=File** command in the console program, or simply **update** and **Bacula** will prompt you for the information.

VolumeNa	Media	VolSta	VolByte	LastWritte	VolRet	Rec
File0001	File	Full	4190055	2002-05-25	14400	0
File0002	File	Full	1897236	2002-05-26	14400	1
File0003	File	Full	1896460	2002-05-26	14400	1
File0004	File	Full	1896460	2002-05-26	14400	1
File0005	File	Full	1896460	2002-05-26	14400	1
File0006	File	Full	1896460	2002-05-26	14400	1
File0007	File	Purged	1896466	2002-05-26	14400	1

In this case, **File0001** will never be automatically recycled. The same effect can be achieved by setting the Volume Status to Read-Only.

As you have noted, the Volume Status (VolStatus) column in the catalog database contains the current status of the Volume, which is normally maintained automatically by **Bacula**. To give you an idea of some of the values it can take during the life cycle of a Volume, here is a picture of the process.

```
A typical volume life cycle is like this:
                because job count or size limit exceeded
    Append ------------------------------------> Used/Full

        | First Job writes to      Retention time passed |
        | the volume                 and recycling takes |
        |                                        place |
        |                                              v
        Recycled <---------------------------------------- Purged
                   Volume is selected for reuse
```

4.2 Pruning

As **Bacula** writes files to a Volume, it keeps a list of files, jobs, and volumes in a database called the Catalog. Among other things, the database helps **Bacula** to decide which files to back up in an incremental or differential backup, and helps you locate files on past backups when you want to restore something. However, the Catalog will grow larger and larger as time goes on, and eventually it can become unacceptably large.

Bacula 's process for removing entries from the Catalog is called Pruning.

The default is Automatic Pruning, which means that once a Job record reaches a certain age (e.g. 30 days old) and a pruning occurs, it will be removed from the Catalog. Note that Job records that are required for current restore won't be removed automatically, and File records are needed for VirtualFull and Accurate backups. Once a job has been pruned, you can still restore it from the backup Volume, provided that the Volume has not been recycled, but one additional step is required: scanning the volume with **bscan**. The alternative to Automatic Pruning is Manual Pruning, in which you explicitly tell **Bacula** to erase the catalog entries for a volume. You'd usually do this when you want to reuse a **Bacula** volume, because there's no point in keeping a list of files that USED TO BE on a volume. Or, if the catalog is starting to get too big, you could prune the oldest jobs to save space. Manual pruning is done with the prune command.

For many deployments, Automatic Pruning is usually fine and does not interfere with normal operations. However, with larger deployments with several millions of files, pruning can take a long time, and can possibly hold up other Jobs.

Additionally, there may be race conditions when Volume pruning is done while backup Jobs are running. This is especially true in large environments, and even more so if Volume truncation is being performed.

The recommended solution for medium and large environment (more than 50 Clients and millions of files) is to disable automatic pruning everywhere, and perform pruning at a quiet time outside of the backup window. The customer has

to create a specific Director's Storage resource pointing to a free SD device used in the Autochanger (or all the devices). The requirement is to have this specific device pointing back to Autochanger it belongs to by using an Autochanger directive.

There are three main resources, on the Director side, where automatic pruning is configured in Bacula:

- the *Job{}* resource
- the *Client{}* resource
- the *Pool{}* resource

For a *Job{}* resource, there are three directives used to configure automatic prune:

- **Prune Jobs = <yes|no>** (default is **no**)
- **Prune Files = <yes|no>** (default is **no**)
- **Prune Volumes = <yes|no>** (default is **no**)

For *Client{}* and *Job{}* resources, there is only one directive used to configure automatic prune:

- **AutoPrune = <yes|no>** (default is **yes**)

If automatic pruning is disabled in these resources (**AutoPrune = no**) and the directives **Prune Jobs**, **Prune Files** or **Prune Volumes** are set to **yes** in a *Job{}* resource, it will override the settings in the *Client{}* resource (for Jobs and Files) and in the *Pool{}* resource (for Volumes).

Notes

- When a volume is pruned, all the Jobs and files associated with this volume are pruned as well. When a Job is pruned, all the files associated with the Job are also pruned. If you have your Job and file retentions greater than your Volume retention, your Jobs and files will be pruned as soon as the Volumes they are stored on is pruned.
- The pruning process is closely related to retention periods configuration. Bacula will only prune Jobs, files or Volumes whose their retention time has expired. There is one exception: a Volume can be pruned and reused if there are no more Jobs and files associated to it in the Catalog, even if the `volume retention time` has not expired. This is usually desirable, provided job retention times match actual requirements. Please check BEPPSchedulesAndRetentions section of this guide for retention period recommendations.
- If a single *Pool{}* or *Client{}* resource has an `Autoprune = yes` set, automatic pruning is activated.

Disabling Automatic Pruning

Pruning by default occurs at the end of a Job. When doing some tests or if you only have few jobs a day, this could be fine. But as soon as the number of jobs is growing, you would prefer to manage pruning another way to let your Catalog do its best for the backup jobs, not for the database administrative tasks.

In order to disable automatic pruning of Jobs and files, you should:

- In *Job{}* resources, avoid the usage of **Prune Jobs = yes** and **Prune Files = yes**.
- In all *Client{}* resources, configure **AutoPrune = no**.

In such a situation, you will configure your clients not to activate the pruning algorithm, using the "Auto Prune" Directive such as the following:

```
Client {
  Name = client-fd
  Address = bacula.example.com
```

(continues on next page)

(continued from previous page)

```
  FDPort = 9102
  Catalog = Catalog
  Password = "do-you-want-a-very-strong-password?"
  File Retention = 15 days
  Job Retention = 50 days
  AutoPrune = no
       # No automatic pruning at the end of a job for this client
}
```

In order to disable automatic pruning of volumes, you should:

- In the *Job{}* resources, avoid the usage of **Prune Volumes = yes**.
- In all *Pool{}* resources, configure **AutoPrune = no**.

Once your Jobs, Clients and Pools have all been edited, you must tell the Director to reload its configuration.

Run the following command to validate your modifications

```
/opt/bacula/bin/bacula-dir -t -u bacula -g bacula
```

Then reload the configuration from **bconsole** with:

```
reload
```

After disabling automatic pruning for Jobs, files and Volumes in all the defined resources, Bacula will no longer perform any automatic pruning. As you might suspect, with no automatic pruning, the Catalog database would grow forever, so we need to perform pruning manually or in a more automated way by using an *Admin* type Job.

Automated Scheduled Pruning When AutoPrune=no

If you don't do anything more, your Catalog will grow infinitely. To keep it at its best, you should define an "Admin" job, like the following:

```
Job {
   Name = "admin-manual-pruning"
   Type = Admin
   JobDefs = "DefaultJob"
   RunScript {
      Runs When = Before
      # below command relies on proper PATH!
      Command = "/bin/sh -c \"echo prune expired volume yes\" | bconsole"
      Runs On Client = no
   }
   Schedule = s-Prune
}
```

As a Bacula volume can contain one or more jobs (or parts of jobs) and a job contains one or more files, the pruning process will have side effects:

- when pruning a Volume, all the jobs related to the Volume are pruned
- when pruning a Job, all the files related to this jobs are pruned

That is the reason why you should have the following:

$$R_f \leqslant R_j \leqslant R_v$$

Where:

- R_f is the File retention period
- R_j is the Job retention period
- R_v is the Volume retention period

In some cases, you might want to disable any form of pruning/purging for a specific Job without changing the configuration of the Client, Job or Pool resources.

It is possible with the **update jobid=x prune=no** bconsole command to update the **Prune** catalog attribute for the given Job.

If the **Prune** Job's catalog attribute is **no** (2 in the catalog) or **files** (1 in the catalog), Bacula will not be able to purge the job record (or the file records) and recycle the associated volumes.

To enable back the regular Job pruning algorithm, use **update jobid=x prune=yes** bconsole command.

To disable the pruning/purging algorithm at the Job resource level, use the **PruneJobs=never** or **PruneFiles=never** directives.

```
Job {
   Name = "BackupCatalog"
   Type = Backup
   JobDefs = "DefaultJob"
   PruneFiles = never
   PruneJobs = never   # implied by PruneFiles = never
}
```

Should you wish to have volumes pruned for specific pools, you can do so by using separated `prune` commands for the different pools, such as the following:

```
echo "prune expired volume pool=Pool_1 yes" | /opt/bacula/bin/bconsole > /tmp/prunePool_
↪1.log
echo "prune expired volume pool=Pool_2 yes" | /opt/bacula/bin/bconsole > /tmp/prunePool_
↪2.log
echo "prune expired volume pool=Pool_3 yes" | /opt/bacula/bin/bconsole > /tmp/prunePool_
↪3.log
```

Pruning Directives

There are three pruning durations. All apply to catalog database Jobs and Files records and not to the actual data in a Volume. The pruning (or retention) durations are for: Volumes (Jobs and Files records in the Volume), Jobs (Job records), and Files (File records). The durations inter-depend a bit because if **Bacula** prunes a Volume, it automatically removes all the Job records, and, consequently, all the File records. Also when a Job record is pruned, all the File records for that Job are also pruned (deleted) from the catalog.

Having the File records in the database means that you can examine all the files backed up for a particular Job. They take the most space in the catalog (probably 90-95% of the total). When the File records are pruned, the Job records can remain, and you can still examine what Jobs ran, but not the details of the Files backed up. In addition, without the File records, you cannot use the Console **restore** command to restore specific files.

When a Job record is pruned, the Volume (Media record) for that Job can still remain in the database, and if you do a **list volumes**, you will see the volume information, but the Job records (and its File records) will no longer be available.

In each case, pruning removes information about where older files are, but it also prevents the catalog from growing to be too large. You choose the retention periods in function of how many files you are backing up and the time periods you want to keep those records online, and the size of the database. You can always re-insert the records (with 98% of the original data) by using **bscan** to scan in a whole Volume or any part of the volume that you want.

By setting **AutoPrune** to **yes** you will permit **Bacula** to automatically prune Volumes in the Pool when a Job needs a Volume. Volume pruning means removing Jobs and Files records from the catalog. It does not shrink the size of the Volume or affect the Volume data until the Volume gets overwritten. When a Job requests a volume and there are no Volumes with Volume Status , or available, **Bacula** will begin volume pruning. This means that all Jobs that are older than the **VolumeRetention** period will be pruned from every Volume that has Volume Status or and has Recycle set to **yes**. Pruning consists of deleting the corresponding Job, File, and JobMedia records from the catalog database. No change to the physical data on the Volume occurs during the pruning process. When all files are pruned from a Volume (i.e. no records in the catalog), the Volume will be marked as implying that no Jobs remain on the volume. The Pool records that control the pruning are described below.

AutoPrune = <yes|no> If AutoPrune is set to **yes******(default), **Bacula** will automatically apply the Volume retention period when running a Job and it needs a new Volume but no appendable volumes are available. At that point, **Bacula** will prune all Volumes that can be pruned (i.e. AutoPrune set) in an attempt to find a usable volume. If during the autoprune, all files are pruned from the Volume, it will be marked with VolStatus . The default is **yes**. Note, that although the File and Job records may be pruned from the catalog, a Volume will be marked Purged (and hence ready for recycling) if the Volume status is Append, Full, Used, or Error. If the Volume has another status, such as Archive, Read-Only, Disabled, Busy, or Cleaning, the Volume status will not be changed to Purged.

Volume Retention = <time-period-specification> The Volume Retention record defines the length of time that **Bacula** will guarantee that the Volume is not reused (i.e. recycled). As long as a Volume has the status **Append**, it will not be recycled even though the retention period may have expired, until the status is changed to some other status such as Full, Used, Purged, ...

The retention period when applied starts at the time of the last write to the Volume. For a Volume to recycled, the volume must not have a status of **Append** and the retention period must have expired. Even in that case, the Volume will not be recycled if any other appendable Volume is available for the Job to complete. As noted above, the Volume status will be marked **Purged** by Bacula prior to it being recycled.

Note, when all the Job records that are on the Volume have been removed, the Volume will be marked (i.e. it has no more valid Jobs stored on it), and the Volume may be recycled even if the **Volume Retention** period has not expired.

When this time period expires, and if **AutoPrune** is set to **yes**, and a new Volume is needed, but no appendable Volume is available, **Bacula** will prune (remove) Job records that are older than the specified Volume Retention period even if the Job or File retention has not been reached. Normally this should not happen since the Volume Retention period should always be set greater than the Job Retention period, which should be greater than the File Retention period.

The Volume Retention period takes precedence over any Job Retention period you have specified in the Client resource. It should also be noted, that the Volume Retention period is obtained by reading the Catalog Database Media record rather than the Pool resource record. This means that if you change the VolumeRetention in the Pool resource record (in bacula-dir.conf), you must ensure that the corresponding change is made in the catalog by using the **update pool** command. Doing so will insure that any new Volumes will be created with the changed Volume Retention period. Any existing Volumes will have their own copy of the Volume Retention period that can only be changed on a Volume by Volume basis using the **update volume** command.

When all Job catalog entries are removed from the volume, its VolStatus is set to . The files remain physically on the Volume until the volume is overwritten.

Retention periods are specified in seconds, minutes, hours, days, weeks, months, quarters, or years on the record. See the Configuration chapter of this manual for additional details of time specification.

The default Volume Retention period is 1 year.

Recycle = <yes|no> This statement tells **Bacula** whether or not the particular Volume can be recycled (i.e. rewritten). If Recycle is set to **no**, then even if **Bacula** prunes all the Jobs on the volume and it is marked , it will not consider the volume for recycling. If Recycle is set to **yes** (the default) and all Jobs have been pruned, the volume status will be set to and the volume may then be reused when another volume is needed. If the volume is reused, it is relabeled with the same Volume Name, however all previous data will be lost.

It is also possible to "force" pruning of all Volumes in the Pool associated with a Job by adding **Prune Files=yes** to the Job resource.

4.3 Pruning and Recycling Example

Perhaps the best way to understand the various resource records that come into play during automatic pruning and recycling is to run a Job that goes through the whole cycle. If you add the following resources to your Director's configuration file:

```
Schedule {
    Name = "30 minute cycle"
    Run = Level=Full Pool=File Messages=Standard Storage=File
    hourly at 0:05
    Run = Level=Full Pool=File Messages=Standard Storage=File
    hourly at 0:35
}
Job {
    Name = "Filetest"
    Type = Backup
    Level = Full
    Client=XXXXXXXXXX
    Fileset="Test Files"
    Messages = Standard
    Storage = File
    Pool = File
    Schedule = "30 minute cycle"
}
# Definition of file storage device
Storage {
    Name = File
    Address = XXXXXXXXXXX
    SDPort = 9103
    Password = XXXXXXXXXXXXX
    Device = FileStorage
    Media Type = File
}
Fileset {
    Name = "File Set"
    Include {
        Options { signature=MD5 }
        File = fffffffffffffff
    }
    Exclude { File=*.o }
}
Pool {
    Name = File
    Use Volume Once = yes
```

(continues on next page)

(continued from previous page)

```
    Pool Type = Backup
    LabelFormat = "File"
    AutoPrune = yes
    VolumeRetention = 4h
    Maximum Volumes = 12
    Recycle = yes
}
```

Where you will need to replace the **ffffffff**'s by the appropriate files to be saved for your configuration. For the Fileset Include, choose a directory that has one or two megabytes maximum since there will probably be approximately eight copies of the directory that **Bacula** will cycle through.

In addition, you will need to add the following to your Storage daemon's configuration file:

```
Device {
    Name = FileStorage
    Media Type = File
    Archive Device = /tmp
    LabelMedia = yes;
    Random Access = Yes;
    AutomaticMount = yes;
    RemovableMedia = no;
    AlwaysOpen = no;
}
```

With the above resources, **Bacula** will start a Job every half hour that saves a copy of the directory you chose to /tmp/File0001 ... /tmp/File0012. After 4 hours, **Bacula** will start recycling the backup Volumes (/tmp/File0001 ...). You should see this happening in the output produced. **Bacula** will automatically create the Volumes (Files) the first time it uses them.

To turn it off, either delete all the resources you've added, or simply comment out the record in the resource.

5 PostgreSQL Catalog Administration

5.1 Catalog Backup

The default backup script is doing an SQL dump. You can change this to other more advanced backup methods supported by PostgreSQL, see https://www.postgresql.org/docs/current/continuous-archiving.html for further suggestions. Make sure to test the full Catalog restore from time to time.

The `pg_dump` tool can run concurrently with backup jobs but we advise starting the Catalog Backup job when no other jobs are running. There are two reasons for this. Running jobs have a specific status set in the Catalog, and when Catalog is restored Director checks it and marks those jobs as failed. Secondly, jobs running while the Catalog is backed up will continue writing to the volumes, and the Catalog dump will not hold all the information backup volumes keep. Bscan is a tool one can use to recreate Catalog records from backup volumes.

5.2 Cleaning up Catalog Database

There are two tables in the Catalog database that occupy most of the space. The File table stores records of all the files that were backed up, and the Path table stores records of all the directory paths. The Path table is shared between all clients and jobs, while the File table will store all files backed up by each backup job.

When files are stored or read the Path table is loaded into memory, and if it grows too much it may affect the performance of the database server.

Usually, Bacula will prune the records from the Catalog when backup jobs are out of retention. Pruning jobs and related files can leave orphaned records. To avoid having too many orphans it is advised to use the dbcheck program periodically to clean the Catalog.

5.3 Vacuum and Analyze

When a record is deleted or updated PostgreSQL doesn't remove the old row from the table but marks it as a dead tuple. Daily operations and orphan removal specifically leave dead tuples in a table. For the database to reuse the table rows occupied by dead tuples it is required to vacuum a table. Regular vacuum ensures the database files do not grow out of proportion keeping dead tuples on disk.

Analyze collects the statistics about column values in tables. These statistics are used to find the best query plan. Rows manipulation changes statistics so regular analyze ensures optimal query performance.

Usually, it would be sufficient to let autovacuum and autoanalyze to deal with vacuuming and statistics. But for large tables, the default settings may be inadequate. Automatic vacuum is controlled by two configuration options with the below defaults.

```
autovacuum_vacuum_threshold = 50
autovacuum_vacuum_scale_factor = 0.2
```

The autovacuum is triggered when:

```
pg_stat_user_tables.n_dead_tup >
(pg_class.reltuples x autovacuum_vacuum_scale_factor)
+ autovacuum_vacuum_threshold
```

Analyze is run after the autovacuum automatically and is similarly controlled with:

```
autovacuum_analyze_threshold = 50
autovacuum_analyze_scale_factor = 0.1
```

Where `autovacuum_analyze_scale_factor` specifies a fraction of the table that is changed before autoanalyze is triggered.

As the number of files kept in the Catalog grows the gap between subsequent autovacuum/autoanalyze executions grows too which results in longer runtimes and possibly innability to run autovacuum/autoanalyze at all. Then we advise running it manually on daily basis and especially after pruning operations. You can schedule this task with `cron` or use the following Admin job during non-production hours (jobs can run concurrently).

```
Job {
  Name = "VACUUM"
  Type = Admin
  JobDefs = DefaultJob
  Runscript {
    Command = "vacuumdb -a -z -q"
    RunsOnClient = no
```

(continues on next page)

(continued from previous page)

```
    RunsWhen = Before
  }
}
```

To avoid using `trust` in the `pg_hba.conf`, you may want to use the `.pgpass` file (with `0600` permission bits) in the `bacula` user's home directory.

```
% ls -l /opt/bacula/.pgpass
-rw------- 1 bacula 201 32 Feb  3 10:19 /opt/bacula/.pgpass

% cat /opt/bacula/.pgpass
*:*:bacula:bacula:yourpassword
```

Note: For large Catalogs, we also advise not to rely on automatic pruning but to schedule maintenance jobs outside of the backup window to deal with records pruning. Please see autopruning for more information. If you run pruning and vacuum in a single Admin job, please take note Bacula cannot guarantee the order of execution for RunScript sections. If there is a need to execute tasks in a set order, please use a shell script and run the script from the Admin job.

A different approach would be to tweak settings for the File table to ensure vacuum and analyze is run automatically. Eliminating `autovacuum_vacuum_scale_factor` from the equation above allows triggering autovacuum as soon as `autovacuum_vacuum_threshold` is crossed. It is important to set the `autovacuum_vacuum_threshold` to a value that would enable vacuum runs automatically after pruning.

For example, if 1 million file records are pruned or updated daily the following settings seem appropriate.

```
ALTER TABLE file SET (autovacuum_vacuum_scale_factor = 0,
autovacuum_vacuum_threshold = 1000000,
autovacuum_analyze_scale_factor = 0,
autovacuum_analyze_threshold = 250000,
autovacuum_vacuum_cost_delay = 0);
```

5.4 dbcheck

dbcheck is a simple program that will search for logical inconsistencies in the Bacula tables in your database, and optionally fix them. It is a database maintenance routine, in the sense that it can detect and remove unused rows, but it is not a database repair routine. To repair a database, see the tools furnished by the database vendor. Normally dbcheck should never need to be run, but if Bacula has crashed or you have a lot of Clients, Pools, or Jobs that you have removed, it could be useful.

The dbcheck program can be found in the /opt/bacula/bin directory.

It is called:

```
Usage: dbcheck [-c config ] [-B] [-C catalog name] [-d debug_level]
  <working-directory> <bacula-database> <user> <password> [<dbhost>] [<dbport>]
       -b              batch mode
       -C              catalog name in the director conf file
       -c              Director conf filename
       -B              print catalog configuration and exit
       -d <nn>         set debug level to <nn>
       -dt             print timestamp in debug output
```

(continues on next page)

(continued from previous page)

```
       -f              fix inconsistencies
       -v              verbose
       -?              print this message
```

If the -B option is specified, dbcheck will print out catalog information in a simple text based format. This is useful to backup it in a secure way.

```
$ dbcheck -B
catalog=MyCatalog
db_type=SQLite
db_name=regress
db_driver=
db_user=regress
db_password=
db_address=
db_port=0
db_socket=
```

If the -c option is given with the Director's conf file, there is no need to enter any of the command line arguments, in particular the working directory as dbcheck will read them from the file.

If the -f option is specified, dbcheck will repair (fix) the inconsistencies it finds. Otherwise, it will report only.

If the -b option is specified, dbcheck will run in batch mode, and it will proceed to examine and fix (if -f is set) all programmed inconsistency checks. If the -b option is not specified, dbcheck will enter interactive mode and prompt with the following:

```
Hello, this is the database check/correct program.
Please select the function you want to perform.
  1) Toggle modify database flag
  2) Toggle verbose flag
  3) Repair bad Filename records
  4) Repair bad Path records
  5) Eliminate duplicate Filename records
  6) Eliminate duplicate Path records
  7) Eliminate orphaned Jobmedia records
  8) Eliminate orphaned File records
  9) Eliminate orphaned Path records
 10) Eliminate orphaned Filename records
 11) Eliminate orphaned Fileset records
 12) Eliminate orphaned Client records
 13) Eliminate orphaned Job records
 14) Eliminate all Admin records
 15) Eliminate all Restore records
 16) All (3-15)
 17) Quit
Select function number:
```

By entering 1 or 2, you can toggle the modify database flag (-f option) and the verbose flag (-v). It can be helpful and reassuring to turn off the modify database flag, then select one or more of the consistency checks (items 3 through 9) to see what will be done, then toggle the modify flag on and re-run the check.

The inconsistencies examined are the following:

- Duplicate filename records. This can happen if you accidentally run two copies of Bacula at the same time, and

they are both adding filenames simultaneously. It is a rare occurrence, but will create an inconsistent database. If this is the case, you will receive error messages during Jobs warning of duplicate database records. If you are not getting these error messages, there is no reason to run this check.

- Repair bad Filename records. This checks and corrects filenames that have a trailing slash. They should not.
- Repair bad Path records. This checks and corrects path names that do not have a trailing slash. They should.
- Duplicate path records. This can happen if you accidentally run two copies of Bacula at the same time, and they are both adding filenames simultaneously. It is a rare occurrence, but will create an inconsistent database. See the item above for why this occurs and how you know it is happening.
- Orphaned JobMedia records. This happens when a Job record is deleted (perhaps by a user issued SQL statement), but the corresponding JobMedia record (one for each Volume used in the Job) was not deleted. Normally, this should not happen, and even if it does, these records generally do not take much space in your database. However, by running this check, you can eliminate any such orphans.
- Orphaned File records. This happens when a Job record is deleted (perhaps by a user issued SQL statement), but the corresponding File record (one for each Volume used in the Job) was not deleted. Note, searching for these records can be very time consuming (i.e. it may take hours) for a large database. Normally this should not happen as Bacula takes care to prevent it. Just the same, this check can remove any orphaned File records. It is recommended that you run this once a year since orphaned File records can take a large amount of space in your database. You might want to ensure that you have indexes on JobId, FilenameId, and PathId for the File table in your catalog before running this command.
- Orphaned Path records. This condition happens any time a directory is deleted from your system and all associated Job records have been purged. During standard purging (or pruning) of Job records, Bacula does not check for orphaned Path records. As a consequence, over a period of time, old unused Path records will tend to accumulate and use space in your database. This check will eliminate them. It is recommended that you run this check at least once a year.
- Orphaned Filename records. This condition happens any time a file is deleted from your system and all associated Job records have been purged. This can happen quite frequently as there are quite a large number of files that are created and then deleted. In addition, if you do a system update or delete an entire directory, there can be a very large number of Filename records that remain in the catalog but are no longer used.

During standard purging (or pruning) of Job records, Bacula does not check for orphaned Filename records. As a consequence, over a period of time, old unused Filename records will accumulate and use space in your database. This check will eliminate them. It is strongly recommended that you run this check at least once a year, and for large database (more than 200 Megabytes), it is probably better to run this once every 6 months.

- Orphaned Client records. These records can remain in the database long after you have removed a client.
- Orphaned Job records. If no client is defined for a job or you do not run a job for a long time, you can accumulate old job records. This option allow you to remove jobs that are not attached to any client (and thus useless).
- All Admin records. This command will remove all Admin records, regardless of their age.
- All Restore records. This command will remove all Restore records, regardless of their age.

If you are using MySQL, dbcheck will ask you if you want to create temporary indexes to speed up orphaned Path and Filename elimination.

Mostly for PostgreSQL users, we provide a pure SQL script dbcheck replacement in examples/database/dbcheck.sql that works with global queries instead of many small queries like dbcheck. Execution instructions are at the top of the script and you will need to type COMMIT at the end to validate modifications.

If you are using bweb or brestore, don't eliminate orphaned Path, else you will have to rebuild brestore_pathvisibility and brestore_pathhierarchy indexes.

By the way, I personally run dbcheck only where I have messed up my database due to a bug in developing Bacula code, so normally you should never need to run dbcheck in spite of the recommendations given above, which are given

so that users don't waste their time running dbcheck too often.

6 Tape/Volume Management

This document describes utility programs written to aid **Bacula** users and developers in dealing with Volumes external to **Bacula**.

6.1 Requirements

- Each of the following programs requires a **valid Storage Daemon configuration file** (actually, the only part of the configuration file that these programs need is the **Device** resource definitions). This permits the programs to find the configuration parameters for your archive device (generally a tape drive). By default, they read **bacula-sd.conf** in the current directory, but you may specify a different configuration file using the **-c** option.
- Each of these programs require a **device-name** where the Volume can be found. In the case of a tape, this is the physical device name such as **/dev/nst0** or **/dev/rmt/0ubn** depending on your system. For the program to work, it must find the identical name in the Device resource of the configuration file. See below for specifying Volume names.

Note: If you have **Bacula** running and you want to use one of these programs, you will either need to stop the Storage Daemon, or `unmount` any tape drive you want to use, otherwise the drive will be **busy** because **Bacula** is using it.

- If you are attempting to read or write an archive file rather than a tape, the **device-name** should be the full path to the archive location including the filename. The filename (last part of the specification) will be stripped and used as the Volume name, and the path (first part before the filename) must have the same entry in the configuration file. So, the path is equivalent to the archive device name, and the filename is equivalent to the volume name.
- In general, you must specify the Volume name to each of the programs below (with the exception of `btape`). The best method to do so is to specify a **bootstrap** file on the command line with the **-b** option. As part of the bootstrap file, you will then specify the Volume name or Volume names if more than one volume is needed. For example, suppose you want to read tapes **tape1** and **tape2**. First construct a **bootstrap** file named say, **list.bsr** which contains:

```
Volume=test1|test2
```

where each Volume is separated by a vertical bar. Then simply use:

```
./bls -b list.bsr /dev/nst0
```

In the case of **Bacula** Volumes that are on files, you may simply append volumes as follows:

```
./bls /tmp/test1\|test2
```

where the backslash () was necessary as a shell escape to permit entering the vertical bar (|).

And finally, if you feel that specifying a Volume name is a bit complicated with a bootstrap file, you can use the **-V** option (on all programs except `bcopy`) to specify one or more Volume names separated by the vertical bar (|). For example,

```
./bls -V Vol001 /dev/nst0
```

You may also specify an asterisk (*) to indicate that the program should accept any volume. For example:


```
./bls -V\* /dev/nst0
```

6.2 bls

bls can be used to do an ls type listing of a **Bacula** tape or file. It is called:

```
Usage: bls [options] <device-name>
-b <file> specify a bootstrap file
-c <file> specify a config file
-d <level> specify debug level
-e <file> exclude list
-i <file> include list
-j list jobs
-k list blocks
(no j or k option) list saved files
-L dump label
-p proceed inspite of errors
-v be verbose
-V specify Volume names (separated by |)
-E Check records to detect errors
-? print this message
```

For example, to list the contents of a tape:

```
./bls -V Volume-name /dev/nst0
```

Or to list the contents of a file:

```
./bls /tmp/Volume-name
```

or

```
./bls -V Volume-name /tmp
```

Note that, in the case of a file, the Volume name becomes the filename, so in the above example, you will replace the **Volume-name** with the name of the volume (file) you wrote.

Normally if no options are specified, bls will produce the equivalent output to the ls -l command for each file on the tape. Using other options listed above, it is possible to display only the Job records, only the tape blocks, etc. For example:

```
./bls /tmp/File002
bls: butil.c:148 Using device: /tmp
drwxrwxr-x 3 k k 4096 02-10-19 21:08 /home/kern/bacula/k/src/dird/
drwxrwxr-x 2 k k 4096 02-10-10 18:59 /home/kern/bacula/k/src/dird/CVS/
-rw-rw-r-- 1 k k 54 02-07-06 18:02 /home/kern/bacula/k/src/dird/CVS/Root
-rw-rw-r-- 1 k k 16 02-07-06 18:02 /home/kern/bacula/k/src/dird/CVS/Repository
-rw-rw-r-- 1 k k 1783 02-10-10 18:59 /home/kern/bacula/k/src/dird/CVS/Entries
-rw-rw-r-- 1 k k 97506 02-10-18 21:07 /home/kern/bacula/k/src/dird/Makefile
-rw-r--r-- 1 k k 3513 02-10-18 21:02 /home/kern/bacula/k/src/dird/Makefile.in
-rw-rw-r-- 1 k k 4669 02-07-06 18:02 /home/kern/bacula/k/src/dird/README-config
-rw-r--r-- 1 k k 4391 02-09-14 16:51 /home/kern/bacula/k/src/dird/authenticate.c
-rw-r--r-- 1 k k 3609 02-07-07 16:41 /home/kern/bacula/k/src/dird/autoprune.c
```

(continues on next page)

(continued from previous page)

```
-rw-rw-r-- 1 k k 4418 02-10-18 21:03 /home/kern/bacula/k/src/dird/bacula-dir.conf
...
-rw-rw-r-- 1 k k 83 02-08-31 19:19 /home/kern/bacula/k/src/dird/.cvsignore
bls: Got EOF on device /tmp
84 files found.
```

Listing Jobs

If you are listing a Volume to determine what Jobs to restore, normally the **-j** option provides you with most of what you will need as long as you don't have multiple clients. For example,

```
./bls -j -V Test1 -c stored.conf DDS-4
bls: butil.c:258 Using device: "DDS-4" for reading.
11-Jul 11:54 bls: Ready to read from volume "Test1" on device "DDS-4" (/dev/nst0).
Volume Record: File:blk=0:1 SessId=4 SessTime=1121074625 JobId=0 DataLen=165
Begin Job Session Record: File:blk=0:2 SessId=4 SessTime=1121074625 JobId=1 Level=F␣
↪Type=B
Begin Job Session Record: File:blk=0:3 SessId=5 SessTime=1121074625 JobId=5 Level=F␣
↪Type=B
Begin Job Session Record: File:blk=0:6 SessId=3 SessTime=1121074625 JobId=2 Level=F␣
↪Type=B
Begin Job Session Record: File:blk=0:13 SessId=2 SessTime=1121074625 JobId=4 Level=F␣
↪Type=B
End Job Session Record: File:blk=0:99 SessId=3 SessTime=1121074625 JobId=2 Level=F Type=B
 Files=168 Bytes=1,732,978 Errors=0 Status=T
End Job Session Record: File:blk=0:101 SessId=2 SessTime=1121074625 JobId=4 Level=F␣
↪Type=B
 Files=168 Bytes=1,732,978 Errors=0 Status=T
End Job Session Record: File:blk=0:108 SessId=5 SessTime=1121074625 JobId=5 Level=F␣
↪Type=B
 Files=168 Bytes=1,732,978 Errors=0 Status=T
End Job Session Record: File:blk=0:109 SessId=4 SessTime=1121074625 JobId=1 Level=F␣
↪Type=B
 Files=168 Bytes=1,732,978 Errors=0 Status=T
11-Jul 11:54 bls: End of Volume at file 1 on device "DDS-4" (/dev/nst0), Volume "Test1"
11-Jul 11:54 bls: End of all volumes.
```

shows a full save followed by two incremental saves.

Adding the **-v** option will display virtually all information that is available for each record:

Listing Blocks

Normally, except for debugging purposes, you will not need to list **Bacula** blocks (the "primitive" unit of **Bacula** data on the Volume). However, you can do so with:

```
./bls -k /tmp/File002
bls: butil.c:148 Using device: /tmp
Block: 1 size=64512
Block: 2 size=64512
...
```

(continues on next page)

(continued from previous page)

```
Block: 65 size=64512
Block: 66 size=19195
bls: Got EOF on device /tmp
End of File on device
```

By adding the **-v** option, you can get more information, which can be useful in knowing what sessions were written to the volume:

```
./bls -k -v /tmp/File002
Volume Label:
Id : Bacula 0.9 mortal
VerNo : 10
VolName : File002
PrevVolName :
VolFile : 0
LabelType : VOL_LABEL
LabelSize : 147
PoolName : Default
MediaType : File
PoolType : Backup
HostName :
Date label written: 2002-10-19 at 21:16
Block: 1 blen=64512 First rec FI=VOL_LABEL SessId=1 SessTim=1035062102 Strm=0 rlen=147
Block: 2 blen=64512 First rec FI=6 SessId=1 SessTim=1035062102 Strm=DATA rlen=4087
Block: 3 blen=64512 First rec FI=12 SessId=1 SessTim=1035062102 Strm=DATA rlen=5902
Block: 4 blen=64512 First rec FI=19 SessId=1 SessTim=1035062102 Strm=DATA rlen=28382
...
Block: 65 blen=64512 First rec FI=83 SessId=1 SessTim=1035062102 Strm=DATA rlen=1873
Block: 66 blen=19195 First rec FI=83 SessId=1 SessTim=1035062102 Strm=DATA rlen=2973
bls: Got EOF on device /tmp
End of File on device
```

Armed with the SessionId and the SessionTime, you can extract just about anything.

If you want to know even more, add a second **-v** to the command line to get a dump of every record in every block.

```
./bls -k -v -v /tmp/File002
bls: block.c:79 Dump block 80f8ad0: size=64512 BlkNum=1
Hdrcksum=b1bdfd6d cksum=b1bdfd6d
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=VOL_LABEL Strm=0 len=147 p=80f8b40
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=SOS_LABEL Strm=-7 len=122 p=80f8be7
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=1 Strm=UATTR len=86 p=80f8c75
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=2 Strm=UATTR len=90 p=80f8cdf
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=3 Strm=UATTR len=92 p=80f8d4d
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=3 Strm=DATA len=54 p=80f8dbd
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=3 Strm=MD5 len=16 p=80f8e07bls: block.c:92␣
↪Rec: VId=1 VT=1035062102 FI=4 Strm=UATTR len=98 p=80f8e2b
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=4 Strm=DATA len=16 p=80f8ea1
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=4 Strm=MD5 len=16 p=80f8ec5
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=5 Strm=UATTR len=96 p=80f8ee9
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=5 Strm=DATA len=1783 p=80f8f5d
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=5 Strm=MD5 len=16 p=80f9668
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=6 Strm=UATTR len=95 p=80f968c
```

(continues on next page)

(continued from previous page)

```
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=6 Strm=DATA len=32768 p=80f96ff
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=6 Strm=DATA len=32768 p=8101713
bls: block.c:79 Dump block 80f8ad0: size=64512 BlkNum=2
Hdrcksum=9acc1e7f cksum=9acc1e7f
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=6 Strm=contDATA len=4087 p=80f8b40
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=6 Strm=DATA len=31970 p=80f9b4b
bls: block.c:92 Rec: VId=1 VT=1035062102 FI=6 Strm=MD5 len=16 p=8101841
...
```

Normally, except for deep debugging purposes, you will not need to verify **Bacula** checksum blocks. However, you can do so with:

```
./bls -j -E -V File002 FileStorage
bls: butil.c:297-0 Using device: "FileChgr1-Dev2" for reading.
14-Feb 09:43 bls JobId 0: Ready to read from volume "TestVolume001" on file device
↪"FileChgr1-Dev2" (/tmp/regress/tmp).
Volume Record: File:blk=0:215 SessId=1 SessTime=1423731064 JobId=0 DataLen=180
14-Feb 09:43 bls JobId 0: Error: block_util.c:451 Volume data error at 0:215!
Block checksum mismatch in block=1 len=64512: calc=34589ccc blk=8d8c5921
```

6.3 btape

This program permits a number of elementary tape operations via a tty command interface. It works only with tapes and not with other kinds of **Bacula** storage media (DVD, File, etc). The **test** command, described below, can be very useful for testing older tape drive compatibility problems. Aside from initial testing of tape drive compatibility with **Bacula**, `btape` will be mostly used by developers writing new tape drivers.

`btape` can be dangerous to use with existing **Bacula** tapes because it will relabel a tape or write on the tape if so requested regardless that the tape may contain valuable data, so please be careful and use it only on blank tapes.

To work properly, `btape` needs to read the Storage daemon's configuration file. As a default, it will look for **bacula-sd.conf** in the current directory. If your configuration file is elsewhere, please use the **-c** option to specify where.

The physical device name must be specified on the command line, and this same device name must be present in the Storage daemon's configuration file read by `btape`

```
Usage: btape <options> <device_name>
    -b <file> specify bootstrap file
    -c <file> set configuration file to file
    -d <nn> set debug level to nn
    -p proceed inspite of I/O errors
    -s turn off signals
    -v be verbose
    -? print this message.
```


Using btape to Verify Tape Drive

An important reason for this program is to ensure that a Storage daemon configuration file is defined so that **Bacula** will correctly read and write tapes.

It is highly recommended that you run the `test` command before running your first **Bacula** job to ensure that the parameters you have defined for your storage device (tape drive) will permit **Bacula** to function properly. You only need to mount a blank tape, enter the command, and the output should be reasonably self explanatory.

btape Commands

The full list of commands are:

```
Command     Description
=======     ===========
autochanger test autochanger
bsf                 backspace file
bsr                 backspace record
cap                 list device capabilities
clear               clear tape errors
eod                 go to end of Bacula data for append
eom                 go to the physical end of medium
fill                fill tape, write onto second volume
unfill              read filled tape
fsf                 forward space a file
fsr                 forward space a record
help                print this command
label               write a Bacula label to the tape
load                load a tape
quit                quit btape
rawfill     use write() to fill tape
readlabel   read and print the Bacula tape label
rectest     test record handling functions
rewind              rewind the tape
scan read() tape block by block to EOT and report
scanblocks  Bacula read block by block to EOT and report
speed               report drive speed
status              print tape status
test                General test Bacula tape functions
weof                write an EOF on the tape
wr                  write a single Bacula block
rr                  read a single record
qfill               quick fill command
```

The most useful commands are:

test
: test writing records and EOF marks and reading them back.

fill
: completely fill a volume with records, then write a few records on a second volume, and finally, both volumes will be read back. This command writes blocks containing random data, so your drive will not be able to compress the data, and thus it is a good test of the real physical capacity of your tapes.

readlabel
read and dump the label on a Bacula tape.

cap
list the device capabilities as defined in the configuration file and as perceived by the Storage daemon.

The `readlabel` command can be used to display the details of a Bacula tape label. This can be useful if the physical tape label was lost or damaged.

In the event that you want to relabel a **Bacula**, you can simply use the `label` command which will write over any existing label. However, please note for labeling tapes, we recommend that you use the `label` command in the **Console** program since it will never overwrite a valid **Bacula** tape.

Testing Tape Drive

See TestTapeDriveDeviceBtapeUtility.

To determine the best configuration of your tape drive, you can run the new `speed` command available in the `btape` program.

This command can have the following arguments:

file_size=n
Specify the Maximum File Size for this test (between 1 and 5GB). This counter is in GB.

nb_file=n
Specify the number of file to be written. The amount of data should be greater than your memory (*file_size* * *nb_file*).

skip_zero
This flag permits to skip tests with constant data.

skip_random
This flag permits to skip tests with random data.

skip_raw
This flag permits to skip tests with raw access.

skip_block
This flag permits to skip tests with Bacula block access.

```
*speed file_size=3 skip_raw
btape.c:1078 Test with zero data and bacula block structure.
btape.c:956 Begin writing 3 files of 3.221 GB with blocks of 129024 bytes.
++++++++++++++++++++++++++++++++++++++++
btape.c:604 Wrote 1 EOF to "Drive-0" (/dev/nst0)
btape.c:406 Volume bytes=3.221 GB. Write rate = 44.128 MB/s
...
btape.c:383 Total Volume bytes=9.664 GB. Total Write rate = 43.531 MB/s
btape.c:1090 Test with random data, should give the minimum throughput.
btape.c:956 Begin writing 3 files of 3.221 GB with blocks of 129024 bytes.
++++++++++++++++++++++++++++++++++++++++
btape.c:604 Wrote 1 EOF to "Drive-0" (/dev/nst0)
btape.c:406 Volume bytes=3.221 GB. Write rate = 7.271 MB/s
++++++++++++++++++++++++++++++++++++++++
...
btape.c:383 Total Volume bytes=9.664 GB. Total Write rate = 7.365 MB/s
```


When using compression, the random test will give your the minimum throughput of your drive . The test using constant string will give you the maximum speed of your hardware chain. (cpu, memory, scsi card, cable, drive, tape).

You can change the block size in the Storage Daemon configuration file.

6.4 bscan

The `bscan` program is designed to recreate database (Catalog) records from backup data stored on one or more Volumes. This process is typically necessary in the event of a disaster or when one or more Volumes have been pruned or purged from your Catalog, resulting in the loss of their records, or for Volumes that have been archived.

Important: In the event of a total disaster of the Catalog database, bscan should be used solely to recover jobs that ran after the point-in-time-recovery of the Catalog database. It means that you should first restore the latest backup of the Catalog database, followed by using bscan to recover the jobids that ran after that.

Note: If you perform a scan on Volumes that have been previously purged, you will still be able to restore data from those Volumes.

However, unless you modify the Job and File retention times for the Jobs added through scanning, any subsequent backup Job with the same name will lead to the records being pruned.

The default configuration for recovered Volumes designates their status as Archive. The timestamp for the last write operation on the Volume is recorded as the time of recovery, and the Volume retention period is established at one month. Consequently, the data within the Catalog will become invalid one month after the completion of the `bscan` recovery process.

Below is the general usage of the bscan command, along with the available options:

```
Usage: bscan [ options ] <bacula-archive>
       -b bootstrap      specify a bootstrap file
       -c <file>         specify configuration file
       -d <nn>           set debug level to <nn>
       -dt               print timestamp in debug output
       -m                update media info in database
       -D <driver name>  specify the driver database name (default NULL)
       -n <name>         specify the database name (default bacula)
       -u <user>         specify database user name (default bacula)
       -P <password>     specify database password (default none)
       -h <host>         specify database host (default NULL)
       -t <port>         specify database port (default 0)
       -p                proceed inspite of I/O errors
       -r                list records
       -s                synchronize or store in database
       -S                show scan progress periodically
       -v                verbose
       -V <Volumes>      specify Volume names (separated by |)
       -w <dir>          specify working directory (default from conf file)
       -?                print this message
```

Note: When using *MySQL* or *PostgreSQL*, specifying a working directory is unnecessary, as `bscan` automatically identifies the database locations. However, if your database is secured, you may need to provide the database name (**-n**

option), along with the user name (**-u** option) and/or the password (**-p** option).

Important: For `bscan` to function properly, it requires at least a minimal valid database present. If your database exists but some records have been pruned, you are ready to proceed. However, if your database has been lost or destroyed, you must first ensure that the SQL server (MySQL or PostgreSQL) is operational. Next, you must create the **Bacula** database (typically named bacula), and set up **Bacula** tables using the scripts in the `/opt/bacula/scripts/` directory. This is explained in the Installation chapter. Finally, before scanning into an empty database, you must start and stop the Director with the appropriate `bacula-dir.conf` file so that it can create the Client and Storage records that are not stored on the Volumes. Without these records, the scanning process cannot link the Job records to the proper Client.

The main use cases for bscan are outlined below.

Using bscan to Compare Volume to Existing Catalog

To compare the contents of a Volume to an existing Catalog without changing the Catalog, it is essential to refrain from using the **-m** or the **-s** options in order to avoid modifying the Catalog. It is important to note that not all records in the Catalog are found on the Volumes, which is why we generally advise **against** using **bscan** except for testing purposes. The most effective method to compare the contents of a Volume with those in the Catalog is to use one of the Verify Job options.

```
bscan -c bacula-sd.conf -v -V Vol001\|Vol002 /dev/nst0
```

This command will provide insight into the expected outcomes without making any changes to your Catalog. Be aware that you may need to adjust the path to the Storage Daemon's configuration file, the Volume name, and the name of your Bacula device (tape, disk, dedup, etc.). This command requires reading the entire Volume, which can be time-consuming if it contains a large amount of data; therefore, you may wish to promptly use the command provided below.

Note: If you are writing to a disk file, ensure that you substitute the device name with the path to the directory containing the Volumes, as this must align with the Archive Device specified in the configuration file.

Using bscan to Synchronize Catalog with Volumes

With some care, `bscan` can also be used to synchronize your existing Catalog with a Volume. Although there have been no reported instances of `bscan` causing damage to a Catalog, it is advisable to backup the existing Bacula database prior to executing `bscan`, as it modifies the Catalog. See Compacting Your PostgreSQL Database or Compacting Your MySQL Database for details of making a copy of your database, or Backing Up Your Bacula Database to use the `make_catalog_backup.pl` program provided by Bacula.

For example, if all records associated with one or both of these Volumes are removed from the database, you can utilize the `bscan` tool to recreate the Catalog entries for those Volumes. Following that, you can employ the `restore` command in the Console to retrieve the desired data.

```
bscan -c bacula-sd.conf -v -V Vol001\|Vol002 /dev/nst0
```

This command will provide insight into the expected outcomes without making any changes to your Catalog. Be aware that you may need to adjust the path to the Storage Daemon's configuration file, the Volume name, and the name of your Bacula device (tape, disk, dedup, etc.). This command requires reading the entire Volume, which can be time-consuming if it contains a large amount of data; therefore, you may wish to promptly use the command provided below.

Note: If you are writing to a disk file, ensure that you substitute the device name with the path to the directory containing the Volumes, as this must align with the Archive Device specified in the configuration file.

Then, to actually write or store the records in the Catalog, add the **-s** option as follows:

```
bscan -s -m -c bacula-sd.conf -v -V Vol001\|Vol002 /dev/nst0
```

When writing to the database, if `bscan` finds existing records, it will generally either update them if discrepancies are detected or leave them unchanged. Therefore, if the Volumes being scanned are already partially or fully in the Catalog, the existing data will remain unaffected. Any missing data will simply be added accordingly.

If you have multiple Volumes, it is recommended to scan them using:

```
bscan -s -m -c bacula-sd.conf -v -V Vol001\|Vol002\|Vol003 /dev/nst0
```

Due to the command line length limitation of 511 bytes imposed by `bscan`, if you possess an excessive number of Volumes, it will be necessary to create a bootstrap file manually. See the Bootstrap for bscan article. For instance, the **.bsr** file for the above example might appear as follows:

```
Volume=Vol001
Volume=Vol002
Volume=Vol003
```

Note: `bscan` does not support supplying Volume names on the command line and at the same time in a bootstrap file. Use only one or the other.

It is advisable to always specify the tapes in the order in which they were written. Failing to do so may result in incomplete or improper restore of any Jobs that span multiple Volumes. However, `bscan` can handle scanning tapes that are not sequential. Any incomplete records at the end of the tape will simply be ignored in that case.

While this may be acceptable if you are merely repairing an existing Catalog, it could lead to an incorrect database state if you are creating a new Catalog from scratch. If all necessary Volumes are not included in a single `bscan` command, the tool will be unable to accurately restore records that span across two Volumes. Therefore, it is far more effective to specify two or three Volumes within a single `bscan` command (or in a `.bsr` file) rather than executing `bscan` multiple times, each time with a single Volume.

Using bscan to Recreate Catalog from Volume

This is the mode for which `bscan` is most useful. You can either `bscan` into a freshly created Catalog, or directly into your existing Catalog, provided you have backed up the current Bacula database as previously outlined. It is generally advisable to start with a freshly created Catalog that contains no data. This can be achieved using the `create_bacula_database`, `make_bacula_tables`, and `grant_bacula_privileges` scripts.

Additionally, `bscan` has the capability to recreate and list records, including Restore Objects and Plugin Objects.

Assuming you start with a single Volume named **TestVolume1**, you execute a command such as:

```
./bscan -V TestVolume1 -v -s -m -c bacula-sd.conf /dev/nst0
```

In the case of multiple Volumes, simply add them to the initial Volume, separating each with a vertical bar. It may be necessary to escape the vertical bar with a forward slash to avoid shell interpretation, e.g. **TestVolume1\|TestVolume2**. The **-v** option was added for verbose output, although it can be omitted if desired. The **-s** option instructs `bscan` to

store information in the database. The physical device name, **/dev/nst0** (which corresponds to the ArchiveDevice value in the `bacula-sd.conf`) is specified following all the options.

For example, after performing a Full backup of a directory followed by two Incremental backups, having recovered the Bacula Catalog as described previously, and using the backup Job boostrap file, the following command was executed:

```
./bscan -b bootstrap.bsr -v -s -c bacula-sd.conf /dev/nst0
```

which produced the following output:

```
bscan: bscan.c:182 Using Database: bacula, User: bacula
bscan: bscan.c:673 Created Pool record for Pool: Default
bscan: bscan.c:271 Pool type "Backup" is OK.
bscan: bscan.c:632 Created Media record for Volume: TestVolume1
bscan: bscan.c:298 Media type "DDS-4" is OK.
bscan: bscan.c:307 VOL_LABEL: OK for Volume: TestVolume1
bscan: bscan.c:693 Created Client record for Client: Rufus
bscan: bscan.c:769 Created new JobId=1 record for original JobId=2
bscan: bscan.c:717 Created Fileset record "Kerns Files"
bscan: bscan.c:819 Updated Job termination record for new JobId=1
bscan: bscan.c:905 Created JobMedia record JobId 1, MediaId 1
bscan: Got EOF on device /dev/nst0
bscan: bscan.c:693 Created Client record for Client: Rufus
bscan: bscan.c:769 Created new JobId=2 record for original JobId=3
bscan: bscan.c:708 Fileset "Kerns Files" already exists.
bscan: bscan.c:819 Updated Job termination record for new JobId=2
bscan: bscan.c:905 Created JobMedia record JobId 2, MediaId 1
bscan: Got EOF on device /dev/nst0
bscan: bscan.c:693 Created Client record for Client: Rufus
bscan: bscan.c:769 Created new JobId=3 record for original JobId=4
bscan: bscan.c:708 Fileset "Kerns Files" already exists.
bscan: bscan.c:819 Updated Job termination record for new JobId=3
bscan: bscan.c:905 Created JobMedia record JobId 3, MediaId 1
bscan: Got EOF on device /dev/nst0
bscan: bscan.c:652 Updated Media record at end of Volume: TestVolume1
bscan: bscan.c:428 End of Volume. VolFiles=3 VolBlocks=57 VolBytes=10,027,437
```

It is important to highlight that `bscan` prints a line of output when each major record is created. However, due to the volume of output, it does not print a line for each file record unless the **-v** option is specified two or more times in the command line.

Note: The restore process using `bscan` is not identical to the original creation of the Catalog data. This is because certain data such as Client records and other non-essential data such as Volume reads, Volume mounts, etc. is not stored on the Volume, and thus is not restored by `bscan`. The results of bscanning are, however, perfectly valid, and will permit restoration of any or all the files in the Catalog using the standard **Bacula** console commands.

If you are starting with an empty Catalog and expecting `bscan` to reconstruct it, this may not occur. It is essential to ensure that your `bacula-dir.conf` file is the same as what it previously was, meaning it must contain all the appropriate Client resources so that they will be recreated in your new database **before** running `bscan`. Typically, when the Director starts, it will recreate any missing Client records in the Catalog.

Additionally, even if the Volumes (Media records) are successfully restored in the database, in the case of tape volumes, their inchanger status and slot assignments will not be correctly configured. Consequently, you will need to repair that by using the `update slots` command.

Important: It is crucial to note that instead of using bscanning, one should always prioritize recovering the previous Catalog backup.

In the case of a Job record, the newly assigned JobId will typically differ from the original JobId. For example, for the first JobId above, the new JobId is 1, but the original JobId is 2. This is is not a cause for concern as it is a standard characteristic of database operations. `bscan` will ensure that everything remains organized.

Although `bscan` indicates that a Client record for Client: Rufus was created three times, it was actually only created the first time. This is expected.

You will also notice that it reads an end of file after each Job (Got `EOF` on device...). The final line will present the total statistics for the `bscan`.

If you had included a second **-v** option to the command line, **Bacula** would have been even more verbose, revealing nearly all the details of each Job record it encountered.

Now, if you start **Bacula** and enter a **list jobs** command to the console program, you will get:

```
+-------+----------+-------------------+------+-----+----------+----------+---------+
| JobId | Name     | StartTime         | Type | Lvl | JobFiles | JobBytes | JobStat |
+-------+----------+-------------------+------+-----+----------+----------+---------+
| 1     | kernsave | 2002-10-07 14:59 | B    | F   | 84       | 4180207  | T       |
| 2     | kernsave | 2002-10-07 15:00 | B    | I   | 15       | 2170314  | T       |
| 3     | kernsave | 2002-10-07 15:01 | B    | I   | 33       | 3662184  | T       |
+-------+----------+-------------------+------+-----+----------+----------+---------+
```

which aligns with its previous state prior to recovery and restore process with `bscan`. All Jobs and Files found on the tape have been successfully restored, including the majority of the Media records. The Volume (Media) records restored will be marked as **Achive**, with a retention period of one month, so that they cannot be rewritten without the intervention of an operator.

It is important to acknowledge that `bscan` cannot restore a database to its exact prior condition, as much of the less critical information within the database is not preserved on the Bacula Volume. However, the reconstruction is sufficiently comprehensive, allowing for the execution of `restore` command to yield valid outcomes.

An interesting aspect of restoring a Catalog backup using `bscan` is that the backup occurs while **Bacula** is running and writing to a tape. At the point the Catalog backup is created, the tape that **Bacula** is writing to may contain, for instance, 10 files. Subsequently, after the Catalog backup is completed, there will be 11 files on the Volume **Bacula** is writing to. his discrepancy highlights the difference between the contents of the backed-up Catalog and the actual files present on the tape. If, after restoring a Catalog, an attempt is made to write on the same Volume that was used to backup the Catalog, **Bacula** will detect the difference in the number of files registered in the Catalog compared to what is on the volume, and it will fix the Catalog automatically.

Disaster Recovery

`bscan` can be useful in a disaster recovery scenarios, particularly following the failure of a hard disk. If there is no valid Catalog database backup, it is possible to use `bscan` to recover the Catalog records from the Volumes. In cases where a valid **bootstrap** file is unavailable for the recovery, or if a Volume has been pruned/purged/recycled but not overwritten, `bscan` can assist in reconstructing your database. This reconstructed database can subsequently be used to **restore** your data as usual using Bacula.

For instance, if you have created backups in the Volumes "Vol001" and "Vol002", and all associated records have been removed from the Catalog, you can use `bscan` to restore those records:

```
bscan -c bacula-sd.conf -v -V Vol001\|Vol002 /dev/nst0
```


If all necessary Volumes are not included in a single `bscan` command, the tool will be unable to accurately restore records that span across multiple Volumes. It is more effective to specify multiple volumes in a single `bscan` command (or in a .bsr file) rather than executing `bscan` multiple times.

After bscan

When using `bscan` to enter the contents of the Volume into an existing Catalog, employing the options `-m update media info in database` or `-s synchronize or store in database` will result in the Volume status being set as `Archive`, with the LastWritten time updated to the current time at which `bscan` accessed the Volume. If neither of these two options is used, the Volume status or the LastWritten values are not changed.

To permit Bacula to reuse this Volume and to prune jobs and files contained within it after the retention period has elapsed, it is necessary to manually adjust the volume status to either `Full` or `Used`.

It is possible to modify the volume status (volstatus) by using the bconsole `update` command.

```
update volstatus=Used volume=Vol-0001
```

6.5 bcopy

The `bcopy` program can be used to copy one **Bacula** archive file to another. For example, you may copy a tape to a file, a file to a tape, a file to a file, or a tape to a tape. For tape to tape, you will need two tape drives. (a later version is planned that will buffer it to disk). In the process of making the copy, no record of the information written to the new Volume is stored in the catalog. This means that the new Volume, though it contains valid backup data, cannot be accessed directly from existing catalog entries. If you wish to be able to use the Volume with the Console restore command, for example, you must first `bscan` the new Volume into the catalog.

bcopy Command Options

```
Usage: bcopy [-d debug_level] <input-archive> <output-archive>
    -b bootstrap specify a bootstrap file
    -c <file> specify configuration file
    -dnn set debug level to nn
    -i specify input Volume names (separated by |)
    -o specify output Volume names (separated by |)
    -p proceed inspite of I/O errors
    -v verbose
    -w dir specify working directory (default /tmp)
    -? print this message
```

By using a **bootstrap** file, you can copy parts of a **Bacula** archive file to another archive.

One of the objectives of this program is to be able to recover as much data as possible from a damaged tape. However, the current version does not yet have this feature.

As this is a new program, any feedback on its use would be appreciated. In addition, I only have a single tape drive, so I have never been able to test this program with two tape drives.

6.6 bextract

Normally, you will restore files by running a **Restore** Job from the **Console** program. However, `bextract` can be used to extract a single file or a list of files from a **Bacula** tape or file. In fact, `bextract` can be a useful tool to restore files to an empty system assuming you are able to boot, you have statically linked `bextract` and you have an appropriate **bootstrap** file.

Some of the current limitations of bextract are:

1. It cannot restore access control lists (`ACL`) that have been backed up along with the file data.
2. It cannot restore encrypted files.
3. The command line length is relatively limited, which means that you cannot enter a huge number of volumes. If you need to enter more volumes than the command line supports, please use a bootstrap file (see below).

It is called:

```
Usage: bextract [-d debug_level] <device-name> <directory-to-store-files>
-b <file> specify a bootstrap file
-c <file> specify a Storage configuration file
-dnn set debug level to nn
-dt print timestamp in debug output
-T send debug traces to trace file
-e <file> exclude list
-i <file> include list
-p proceed inspite of I/O errors
-t read data from volume, do not write anything
-v verbose
-V specify Volume names (separated by |)
-? print this message
```

where **device-name** is the Archive Device (raw device name or full filename) of the device to be read, and **directory-to-store-files** is a path prefix to prepend to all the files restored.

Note: On Windows systems, if you specify a prefix of say `d:/tmp`, any file that would have been restored to `c:/My Documents` will be restored to `d:/tmp/My Documents`. That is, the original drive specification will be stripped. If no prefix is specified, the file will be restored to the original drive.

Extracting with Include or Exclude Lists

Using the **-e** option, you can specify a file containing a list of files to be excluded. Wildcards can be used in the exclusion list. This option will normally be used in conjunction with the **-i** option (see below). Both the **-e** and the **-i** options may be specified at the same time as the **-b** option. The bootstrap filters will be applied first, then the include list, then the exclude list.

Likewise, and probably more importantly, with the **-i** option, you can specify a file that contains a list (one file per line) of files and directories to include to be restored. The list must contain the full filename with the path. If you specify a path name only, all files and subdirectories of that path will be restored. If you specify a line containing only the filename (e.g. **my-file.txt**) it probably will not be extracted because you have not specified the full path.

For example, if the file **include-list** contains:

```
/home/kern/bacula
/usr/local/bin
```


Then the command:

```
./bextract -i include-list -V Volume /dev/nst0 /tmp
```

will restore from the **Bacula** archive **/dev/nst0** all files and directories in the backup from `/home/kern/bacula` and from `/usr/local/bin`. The restored files will be placed in a file of the original name under the directory `/tmp` (i.e. `/tmp/home/kern/bacula/...` and `/tmp/usr/local/bin/...`).

Extracting With Bootstrap File

The **-b** option is used to specify a **bootstrap** file containing the information needed to restore precisely the files you want. Specifying a **bootstrap** file is optional but recommended because it gives you the most control over which files will be restored. For more details on the **bootstrap** file, please see **Restoring Files with the Bootstrap File** chapter of this document. Note, you may also use a bootstrap file produced by the `restore` command. For example:

```
./bextract -b bootstrap-file /dev/nst0 /tmp
```

The bootstrap file allows detailed specification of what files you want restored (extracted). You may specify a bootstrap file and include and/or exclude files at the same time. The bootstrap conditions will first be applied, and then each file record seen will be compared to the include and exclude lists.

Extracting From Multiple Volumes

If you wish to extract files that span several Volumes, you can specify the Volume names in the bootstrap file or you may specify the Volume names on the command line by separating them with a vertical bar. See the section above under the `bls` program entitled **Listing Multiple Volumes** for more information. The same techniques apply equally well to the `bextract` program or read the **Bootstrap** chapter of this document.

Test Extraction

If you wish to ensure that all blocks of a volume are valid, you can specify the **-t** flag. A bootstrap might be used with this option. It is also possible to setup and use a Verify Job.

7 Tape Autochanger Usage

7.1 Tape Drives: update slot inventory and labeling tapes

Labelling tapes

If you want to use Tape Autochangers, please consider labelling them by barcode. Normally you can add new Volumes into a “Scratch Pool” where other Pools pick them. You can assign them directly to a given Pool if you prefer.

If you have a tape drive that is idle during the label command, it should be chosen automatically. Assuming that your 1st Drive (drive=0) is busy with backups, you can, in this case, use the 2nd drive of your library . If not chosen automatically, you can select the 2nd drive by using “drive=1” on the command line.

Note: The tapes should be labeled as scratch pool tapes. To read more about configuring pools, click here.

Updating Slots Inventory

When changing tape cartridges in your Library by adding or removing Tapes or moving them around you should issue an `update slots` command within **bconsole**. Please be aware that some Libraries might unload a used cartridge to an arbitrary free slot, if the selected slot for unloading is currently occupied.

```
update slots barcode drive=1 storage=Autochanger-LTO
```

After that Bacula knows where the cartridges are, and the barcode sticker of each individual tape has been read. Now you must do the labelling with an additional command. Please use a non busy drive to label your barcodes.

Sample output

Assuming new tapes are in slots 30,29,28,16,14, you should choose a Pool, Storage and Slot(s) (drive), otherwise you will be prompted interactively.

```
label barcode storage=Autochanger-LTO pool=Scratch slots=30,29,28,16,14 drive=1
```

The **bconsole** `status slots` command gives you a list of your current slots assignments.

```
status slots

 Slot |   Volume Name    |   Status  |     Media Type       |      Pool          |
------+------------------+-----------+----------------------+--------------------|
    1*|           VTL001 |    Append |                  LTO |                VTL |
    2 |                  |           |                      |                    |
```

8 Credentials Management

9 Pool Management

10 Users and Roles Management

