Bacula Regression Testing

Bacula Community Version

This document is intended to provide insight into the considerations and processes required to implement regression testing with the Community Version of Bacula.

Bacula Systems White Paper

Version 1.3, Saturday 15th December, 2018

www.baculasystems.com

Contents

Bacula Regression Testing

If you decided to install **Bacula** from source, and want to be as sure as possible that your self-compiled binaries behave as expected, the regression testing suite included with the source code is the best way to do so. You do not need to be a developer to run the regression scripts.
If you installed **Bacula** from packages, going through the regression tests can be a reasonable way to ensure that everything behaves as expected and can be an important part of the pre-production testing routine.
The tests are shell scripts that drive Bacula through bconsole and then typically compare the input and output with `diff`, so no particular infrastructure beyond what you needed to build **Bacula** and what is part of a typical Unix / Linux system is required.
To get started, we recommend that you create a directory named `bacula`, under which you will put the current source code and the current set of regression scripts. The directory could have any name you like, though.
Note, all the standard regression scripts run as non-root and can be run on the same machine as a production Bacula system.
To create the directory structure for the current trunk code and to checkout the necessary files, do the following (note, we assume you are working in your home directory in a non-root account):

1. Create a directory to be used for testing
2. Check out from the **Bacula** `git` repository into that directory, or use a `tar` archive:
   - **Bacula** source code
   - **Bacula** regression suite

If you use the `git` repository, you can keep the source code and regression suite up-to-date by fetching from the repository.
If you want to test with SQLite and it is not installed on your system, you will need to download the latest depkgs release from Source Forge and unpack it into `bacula/depkgs`, then simply:

```
cd bacula/depkgs
make
```

Note, you can also use PostgreSQL or MySQL for testing; both of them will usually be available from distribution repositories.
There are two different aspects of regression testing: Running the Regression Tests, and writing a Regression Test. We will not go into much detail on the latter, though, as we expect that creating regression test scripts will typically be done by developers of new features of **Bacula**, not users.

1.1 Running the Regression Script

There are almost 200 tests and the number increases over time. Each of these tests checks a particular feature or use case, but in general it's not very interesting to run all those tests one by one. Instead, most of the time the overall picture is most interesting: Do all the tests, or all the ones testing the required features, succeed? The tests also vary in length, some of them require tape drives, and some need root permission. It's also not very interesting to watch tests going on for several hours, so we have prepared a number of **test sets** like the standard set that uses disk volumes and runs under any userid, a small set of tests that write to tape, another set of tests where you must be root to run them.
Each test is self contained in that it initializes to run **Bacula** from scratch (i. e. with a newly created database and an uncluttered configuration). It will also kill any **Bacula** session that is currently running. In addition, it uses ports 8101, 8102, and 8103 so that it does not interfere with a production system.

1.2 Setting the Configuration Parameters

All the regression testing configuration is strictly separated from the **Bacula** source code, so there is nothing you need to change in the source directory.
The very first time you are going to run the regression scripts, you will need to create a custom config file for your system which has to be located in the `regress` directory. We suggest that you start by:

```
cd bacula/regress
cp prototype.conf config
```

Afterwords, you can edit the `config` file directly, which might look similar to what we show in listing **??**
The contents of the configuration file should contain these items:

BACULA_SOURCE should be the full path to the Bacula source code that you wish to test. It will be loaded, configured, compiled, and installed with the "make setup" command, which needs to be done only once each time you change the source code.

EMAIL should be your email address. Please remember to change this to something that works for your site!

SMTP_HOST defines where your SMTP server is.

SQLITE_DIR should be the full path to the `SQLite` package, which must be built before running a Bacula regression test, if you are using `SQLite`. This variable is ignored if you are using `MySQL` or `PostgreSQL`.

WHICHDB is used to configure which database to use. Set it to `--with-postgresql`, `--with-mysql`, `--with-sqlite3=${SQLITE3_DIR}` or `--with-sqlite=${SQLITE_DIR}` as needed.

The advantage of using SQLite is that it is totally independent of any installation you may have running on your system, and there is no special configuration or authorization that must be done to run it. With both `MySQL`

(Illustrative material only)

Listing 1.1: Regression Tests Configuration

```
# Where to get the source to be tested
BACULA_SOURCE="${HOME}/bacula/bacula"

# Where to send email   !!!!! Change me !!!!!!!
EMAIL=your-name@your-domain.com
SMTP_HOST="localhost"

# Full "default" path where to find sqlite (no quotes!)
SQLITE3_DIR=${HOME}/bacula/depkgs/sqlite3
SQLITE_DIR=${HOME}/bacula/depkgs/sqlite

TAPE_DRIVE="/dev/nst0"
# if you don't have an autochanger set AUTOCHANGER to /dev/null
AUTOCHANGER="/dev/sg0"
# For two drive tests -- set to /dev/null if you do not have it
TAPE_DRIVE1="/dev/null"

# This must be the path to the autochanger control program including its ↙
→name
AUTOCHANGER_PATH="/usr/sbin/mtx"

# Set your database here
#WHICHDB="--with-sqlite=${SQLITE_DIR}"
#WHICHDB="--with-sqlite3=${SQLITE3_DIR}"
#WHICHDB="--with-mysql"
WHICHDB="--with-postgresql"

# Set this to "--with-tcp-wrappers" or "--without-tcp-wrappers"
TCPWRAPPERS="--with-tcp-wrappers"

# Set this to "" to disable OpenSSL support, "--with-openssl=yes"
# to enable it, or provide the path to the OpenSSL installation,
# eg "--with-openssl=/usr/local"
OPENSSL="--with-openssl"

# You may put your real host name here, but localhost is valid also
#  and it has the advantage that it works on a non-networked machine
HOST="localhost"
```

and `PostgreSQL`, you must pre-install the packages, initialize them and ensure that you have authorization to access the database and create and delete tables.

TAPE_DRIVE is the full path to your tape drive. The base set of regression tests do not use a tape, so this is only important if you want to run the full tests. Set this to `/dev/null` if you do not have a tape drive.

TAPE_DRIVE1 is the full path to your second tape drive, if you have one. The base set of regression tests do not use a tape, so this is only important if you want to run the full two drive tests. Set this to `/dev/null` if you do not have a second tape drive.

AUTOCHANGER is the name of your autochanger control device. Set this to `/dev/null` if you do not have an autochanger.

AUTOCHANGER_PATH is the full path including the program name for your autochanger control program (normally **`mtx`**). Leave the default value if you do not have one.

TCPWRAPPERS defines whether or not you want the **`./configure`** to be performed with TCP wrappers enabled.

OPENSSL used to enable/disable SSL support for Bacula communications and data encryption.

HOST is the hostname that it will use when building the scripts. The Bacula daemons will be named <HOST>-dir, <HOST>-fd, ... It is also the name of the machine to connect to the daemons by the network. Hence the name should either be your real hostname (with an appropriate DNS or `/etc/hosts` entry) or `localhost` as it is in the default file.

bin is the binary program location.

scripts is the bacula scripts location (where database creation script, autochanger handler, etc. are found).

1.3 Building the Test Bacula

When the configuration is prepared, you can build the `Makefile` used to create the **Bacula** installation for the regression testing suite by entering `./config config` where the second `config` refers to the name of the configuration file containing your system parameters. This will build a `Makefile` from `Makefile.in`, and you should not need to do this again unless you want to change the database or other regression configuration parameter.

1.4 Setting up your SQL engine

If you are using `SQLite` or `SQLite3`, there is nothing more to do; you can simply run the tests as described in the next section.
If you are using `MySQL` or `PostgreSQL`, you will need to establish an account with your database engine for the user name **regress** and you will need to manually create

a database named **regress** that can be used by user name regress, which means you will have to give the user **regress** sufficient permissions to fully use the database named **regress**. There must not be a password required for this access.
You have probably already done this for a production **Bacula** instance with user name and database name **bacula**. If you did not, the manual describes how to do it, and the scripts in `bacula/regress/build/src/cats` called `create_mysql_database`, `create_postgresql_database`, `grant_mysql_privileges`, and `grant_postgresql_privileges` may be of a help to you.
Generally, to do the above, you will need to work with root permissions to be able to create databases and modify permissions within `MySQL` and `PostgreSQL`.

1.5 Running the Disk Only Regression

To run the base set of tests using disk volumes, you would do

```
./do_disk
```

If you are testing on a non-Linux machine several of the tests may not run. In any case, as we add new tests, the number will vary. It will take about 1 hour and you don't need to be root to run these tests. The result should look similar to what is shown in figure **??**
Alternatively to using the **`./do_disk`** script, you can manually do what it essentially does by:

```
make setup
./all-disk-tests
scripts/cleanup
```

The above will first copy the source code within the regression tree (in directory regress/build), configure it, and build it. There should be no errors. If there are, please correct them before continuing. From this point on, as long as you don't change the Bacula source code, you should not need to repeat any of the building steps steps (which **`make`** will do correctly, i. e. it will only build the binaries if the sources have been modified). If you pull down a new version of the source code, simply run **make setup** again.
Once Bacula is built, you can run the basic disk only non-root regression test by entering:

```
make test
```

and the tape tests are run with

```
make full_test
```

and will create output similar to the one shown in figure **??**.

1.6 Other Tests

There are a number of other tests that can be run as well. All the tests are a simple shell script kept in the regress directory. For example the "make test" simply executes **`./all-non-root-tests`**. The other tests, which are invoked by directly running the script, are:

```
Test results
  ===== auto-label-test OK 12:31:33 =====
  ===== backup-bacula-test OK 12:32:32 =====
  ===== bextract-test OK 12:33:27 =====
  ===== bscan-test OK 12:34:47 =====
  ===== bsr-opt-test OK 12:35:46 =====
  ===== compressed-test OK 12:36:52 =====
  ===== compressed-encrypt-test OK 12:38:18 =====
  ===== concurrent-jobs-test OK 12:39:49 =====
  ===== data-encrypt-test OK 12:41:11 =====
  ===== encrypt-bug-test OK 12:42:00 =====
  ===== fifo-test OK 12:43:46 =====
  ===== backup-bacula-fifo OK 12:44:54 =====
  ===== differential-test OK 12:45:36 =====
  ===== four-concurrent-jobs-test OK 12:47:39 =====
  ===== four-jobs-test OK 12:49:22 =====
  ===== incremental-test OK 12:50:38 =====
  ===== query-test OK 12:51:37 =====
  ===== recycle-test OK 12:53:52 =====
  ===== restore2-by-file-test OK 12:54:53 =====
  ===== restore-by-file-test OK 12:55:40 =====
  ===== restore-disk-seek-test OK 12:56:29 =====
  ===== six-vol-test OK 12:57:44 =====
  ===== span-vol-test OK 12:58:52 =====
  ===== sparse-compressed-test OK 13:00:00 =====
  ===== sparse-test OK 13:01:04 =====
  ===== two-jobs-test OK 13:02:39 =====
  ===== two-vol-test OK 13:03:49 =====
  ===== verify-vol-test OK 13:04:56 =====
  ===== weird-files2-test OK 13:05:47 =====
  ===== weird-files-test OK 13:06:33 =====
  ===== migration-job-test OK 13:08:15 =====
  ===== migration-jobspan-test OK 13:09:33 =====
  ===== migration-volume-test OK 13:10:48 =====
  ===== migration-time-test OK 13:12:59 =====
  ===== hardlink-test OK 13:13:50 =====
  ===== two-pool-test OK 13:18:17 =====
  ===== fast-two-pool-test OK 13:24:02 =====
  ===== two-volume-test OK 13:25:06 =====
  ===== incremental-2disk OK 13:25:57 =====
  ===== 2drive-incremental-2disk OK 13:26:53 =====
  ===== scratch-pool-test OK 13:28:01 =====
Total time = 0:57:55 or 3475 secs
```

Figure 1.1: Typical Test Results for Disk Volume Regression Testing

```
Test results

  ===== Bacula tape test OK =====
  ===== Small File Size test OK =====
  ===== restore-by-file-tape test OK =====
  ===== incremental-tape test OK =====
  ===== four-concurrent-jobs-tape OK =====
  ===== four-jobs-tape OK =====
```

Figure 1.2: Tape Test Results

all_non-root-tests All non-tape tests not requiring root permissions. This is the standard set of tests, that in general back up some data, restore it, and finally compare the restored with the original data.

all-root-tests All non-tape tests requiring root permission. These are a relatively small number of tests that require running as root. The amount of data backed up can be quite large. For example, one test backs up /usr, another backs up /etc. One or more of these tests may report errors, which are usually **not critical**.

all-non-root-tape-tests All tape test not requiring root. There are currently three tests: The first two tests use one volume, and the third test requires an autochanger, and uses two volumes. If you don't have an autochanger, then this script will probably produce an error.

all-tape-and-file-tests All tape and file tests not requiring root. This includes almost everything.

1.7 If a Test Fails

If you one or more tests fail, the line output will be similar to:

```
!!!!! concurrent-jobs-test failed!!! !!!!!
```

If you want to determine why the test failed, you will need to rerun the script with the debug output turned on. You do so by defining the environment variable `REGRESS_DEBUG` with commands such as:

```
REGRESS_DEBUG=1
export REGRESS_DEBUG
```

Then from the `regress` directory (all regression scripts assume that you have that as the current directory), enter:

```
tests/test-name
```

where `test-name` should be the name of a test script – for example: `tests/backup-bacula-test`.

1.8 Testing Your Distribution

First, make sure that your configuration uses the same catalog backend as your distribution-provided **Bacula**. Then you can set the `bin` and `scripts` variables in your config file.
Example:

```
bin=/opt/bacula/bin
scripts=/opt/bacula/scripts
```

The `./scripts/prepare-other-loc` will tweak the installed regression testing scripts to use the new binary locations. You will have to run `make setup` to be able to use regression binaries again.

```
$ ./scripts/prepare-other-loc
$ ./tests/backup-bacula-test
...
```

1.9 Running a Single Test

If you wish to run a single test, you can simply do:

```
cd regress
tests/<name-of-test>
```

Remember that, if the source code has been updated, you should check out or download and unpack the current source code and use **make** to set up the regression testing environment before running an individual test.

1.10 Writing a Regression Test

Any developer who implements a major new feature should write a regression test that exercises and validates the new feature. Each regression test is a complete test by itself. It terminates any running Bacula, initializes the database, starts Bacula, then runs the test by using the **bconsole** program.

1.11 Directory Structure

The directory structure of the regression test suite is:

```
  regress                - Makefile, scripts to start tests
    |------ scripts      - Scripts and conf files
    |-------tests        - All test scripts are here
    |
    |- - - - - - - - -  -- All directories below this point are used
    |                       for testing, but are created from the
    |                       above directories and are removed with
    |                       "make distclean"
    |
    |------ bin          - This is the install directory for
    |                        Bacula to be used for testing
    |------ build        - Where the Bacula source tree is built
    |------ tmp          - Most temp files go here
    |------ working      - Bacula working directory
    |------ weird-files  - Weird files used in two of the tests.
```

1.12 Adding a New Test

If you want to write a new regression test, it is best to start with one of the existing test scripts, and modify it to do the new test.
When adding a new test, be extremely careful about adding anything to any of the daemons' configuration files. The reason is that it may change the prompts that are sent to the console. For example, adding a Pool means that the current scripts, which assume that Bacula automatically selects a Pool, will now be presented with a new prompt, so the tests will fail. If you need to enhance the configuration files, consider making your own versions.

1.13 Running a Test Under The Debugger

You can run a test under the debugger (actually run a Bacula daemon under the debugger) by first setting the environment variable `REGRESS_WAIT` with commands such as:

```
REGRESS_WAIT=1
export REGRESS_WAIT
```

Then executing the script. When the script prints the following line:

```
Start Bacula under debugger and enter anything when ready ...
```

You start the Bacula component you want to run under the debugger in a different shell window. For example:

```
cd .../regress/bin
gdb bacula-sd
(possibly set breakpoints, ...)
run -s -f
```

Then enter any character in the window with the above message. An error message will appear saying that the daemon you are debugging is already running, which is correct, so you should simply ignore the error message.

For More Information

For more information on Bacula Enterprise Edition, or any part of the broad Bacula Systems services portfolio, visit www.baculasystems.com.

Rev : 2 V. 1.3
Author(s): ARL,KES