Backup Manager 0.7.7 User Guide

Alexis Sukrieh

1.7 - 14 Apr, 2008

Copyright Notice

Contents

Chapter 1

About this manual

1.1 Scope

Backup Manager is a system tool designed to handle backups. It is written with simplicity in mind.

If you want to handle a couple of tarballs, reading the default configuration file might be enough to understand the main design. On the other hand, if you want to know more about the global design of the program, how to write your own backup methods or even look at some real life examples, this guide is for you.

This document describes the main design of the software and gives information about supported configuration keys. All backup methods are described, with a sample configuration file as illustration. Whenever possible, advices and best practices are given.

This manual also describes every configuration variables supported in the version 0.7.7.

1.2 Version

This document is updated whenever a new release of Backup Manager is published. The current version covers all features and configuration details about version 0.7.7.

The first version of this document was written with the release 0.6 of Backup Manager.

1.3 Authors

The first version of this document was made in late 2005, by Alexis Sukrieh and has been reviewed by Sven Joachim.

While the author of this document has tried hard to avoid typos and other errors, these do still occur. If you discover an error in this manual or if you want to give any comments, suggestions, or criticisms please create a new issue at https://github.com/sukria/Backup-Manager/issues

Chapter 2

Configuration files

Backup Manager's behaviour is defined in configuration files. You can run Backup Manager with different configuration files (at the same time or not). This chapter will cover all the configuration keys supported in version 0.7.7 and will explain their meaning.

2.1 Repository and Archives

Backup Manager stores *archives* it builds in a *repository*. *Archives* are built by using a *backup method*.

2.1.1 The Repository

BM_REPOSITORY_ROOT

Type: string, default: `/var/archives`.

The repository is the place in your filesystem where all archives are stored. This is a particular place for Backup Manager, it will be cleaned during backup sessions: archives older than the authorized lifetime will be purged. If the repository does not exist, it will be created at runtime.

Isolating the repository on a dedicated partition is a good idea. This can prevent the repository from eating all the disk space of the partition. With a bad configuration file, backup sessions can lead to huge archives, for many reasons, so take care.

Example:

```
export BM_REPOSITORY_ROOT="/var/archives"
```

BM_REPOSITORY_SECURE

Type: boolean, default: `true`.

For security reasons, the repository can be accessible by a specific user/group pair. This will prevent the archives from being readable (and writable) by any user in the system. This mode is enabled by default (owned by `root:root`).

To enable this mode, set the configuration key `BM_REPOSITORY_SECURE` to `yes`, then update `BM_REPOSITORY_USER` and `BM_REPOSITORY_GROUP` to your needs.

You can also change the permission of the repository and the archives, that is possible with two configuration variables: `BM_REPOSITORY_CHMOD` and `BM_ARCHIVE_CHMOD`.

Example:

```
export BM_REPOSITORY_SECURE="true"
export BM_REPOSITORY_USER="root"
export BM_REPOSITORY_GROUP="root"
export BM_REPOSITORY_CHMOD="770"
export BM_ARCHIVE_CHMOD="660"
```

2.1.2 Encryption

If you cannot trust the place where you store your archives, you can choose to encrypt them so you are the only one who can read their content. That's a very good idea for archives you plan to upload to some remote place, or even for the archives you want to daily export on removable media.

BM_ENCRYPTION_METHOD

Type: string, default: undefined.

For Backup Manager, encryption is defined in one place in the configuration file. If the variable "BM_ENCRYPTION_METHOD" is not defined, no encryption occurs during the archive build process, if a method is defined there, then any archive built are encrypted through a pipeline with that method.

Be aware that encryption is supported for the methods "mysql", "pipe", "tarball" and "tarball-incremental" but only for those file types: tar, tar.gz, tar.bz2.

The only valid method supported for encrypting archives is "gpg".

Backup Manager will encrypt your archive through a pipeline in order not to write any byte of unencrypted data on the physical media. The encryption will be performed with a command line like the following:

```
<command> | gpg -r "$BM_ENCRYPTION_RECIPIENT" -e > archive.gpg
```

To decrypt an archive built with GPG encryption, you have to be the owner of the private GPG key for which the encryption was made. Then issue the following:

```
$ gpg -d <archive.gpg> > archive
```

GPG will then prompt you for the private key passphrase and will decrypt the content of the archive if the passphrase is valid.

Refer to the GPG documentation for more details of encryption.

BM_ENCRYPTION_RECIPIENT

Type: string, default: undefined.

As explained in the previous section, that variable should contain the GPG recipient for the encryption, eg: your GPG ID.

Examples of valid GPG ID:

```
export BM_ENCRYPTION_RECIPIENT="0x1EE5DD34"
export BM_ENCRYPTION_RECIPIENT="Alexis Sukrieh"
export BM_ENCRYPTION_RECIPIENT="sukria@sukria.net"
```

2.1.3 Archives

Archives are produced by backup methods, they can be virtually anything, but will always be named like the following: `prefix-name-date.filetype`. An archive is a file that contains data, it can be compressed or not, in a binary form or not.

BM_ARCHIVE_STRICTPURGE

Type: boolean, default: `true`.

As explained in the BM_REPOSITORY_ROOT section, every archive built by Backup Manager will be purged when their lifetime expires. In versions prior to 0.7.6, any archive were purged.

You can now choose to purge only the archive built in the scope of the configuration file, that is: archives prefixed with BM_ARCHIVE_PREFIX.

This is useful if you share the same BM_REPOSITORY_ROOT with different instances of Backup Manager that have different purging rules (eg: a BM_REPOSITORY_ROOT shared over NFS for multiple Backup Manager configuration).

Example:

```
export BM_ARCHIVE_STRICTPURGE="true"
```

BM_ARCHIVE_NICE_LEVEL

Type: string, default: `10`.

Backup Manager does handle several archive methods, which can use a lot of ressources (mostly CPU); although this can be acceptable if Backup Manager is run at night, on a always-running server, it can seriously slow-down a desktop computer. Indeed, most of the time, desktop users use anacron to run backup-manager when possible, and most of time this is when the desktop is actually used.

To enhance the desktop-experience when archives are built, you can adjust the niceness used for archive creation with this configuration variable.

To set a low priority to the archive creation processes, use a high number (max: 19). See the manpage of nice for details.

Example:

```
export BM_ARCHIVE_NICE_LEVEL="19" # recommanded for desktop users.
```

BM_ARCHIVE_PURGEDUPS

Type: boolean, default: `true`.

If disk usage matters in your backup strategy, you might find useful to use Backup Manager's duplicates purging feature. When an archive is generated, Backup Manager looks at the previous versions of this archive. If it finds that a previous archive is the same file as the one it has just built, the previous one is replaced by a symlink to the new one. This is useful if you don't want to have the same archive twice in the repository.

Example:

```
export BM_ARCHIVE_PURGEDUPS="true"

host-etc.20051115.tar.gz
host-etc.20051116.tar.gz -> /var/archives/host-etc.20051117.tar.gz
host-etc.20051117.tar.gz
```

BM_ARCHIVE_TTL

Type: integer, default: `5`.

One of the main concepts behind the handling of the repository is to purge deprecated archives automatically. The purge session is always performed when you launch Backup Manager. During this phase, all archives older than the authorized lifetime are dropped.

Since version 0.7.3, Backup Manager purges only files it has created whereas in previous versions, it used to purge also other files within the repository.

Note that when using the incremental method for building archives, Backup Manager will handle differently master backups and incremental ones. The incremental backups will be purged like any other archives (when exceeding the authorized lifetime). On the ohter hand, deprecated master backups won't be purged unless there is a younger master backup in the repository. Then, even with a lifetime set to three days, a master backup will live more than three days, until a newer master backup is built.

Example:

```
export BM_ARCHIVE_TTL="5"
```

BM_REPOSITORY_RECURSIVEPURGE

Type: boolean, default: `false`.

On most setups, all the archives are stored in the top-level directory specified by the configuration key `BM_REPOSITORY_ROOT`. But it can make sense to have subdirectories, for instance to store archives uploaded from other hosts running Backup Manager. In this case, it is possible to ask Backup Manager to purge those directories too, by setting `BM_REPOSITORY_RECURSIVEPURGE` to `true`.

Please note that the `BM_ARCHIVE_TTL` value is global, so if you want to have different lifetimes for some archives, this is not the way to go. In this case you should save them outside `BM_REPOSITORY_ROOT` and write a cron job to do the purge (possibly calling `backup-manager --purge` with an alternate configuration file).

Example:

```
export BM_REPOSITORY_RECURSIVEPURGE="false"
```

BM_ARCHIVE_PREFIX

Type: string, default: `$HOSTNAME`.

This is the prefix used for naming archives.

Example:

```
export BM_ARCHIVE_PREFIX="$HOSTNAME"

# echo $HOSTNAME
ouranos
# ls /var/archives
ouranos-20051123.md5
ouranos-usr-local-src.20051123.tar.gz
ouranos-etc.20051123.tar.gz
```

2.2 Backup Methods

The core feature of Backup Manager is to make archives, for doing this, a *method* is used. Each method can require a set of configuration keys. We will describe here every method supported in the version 0.7.7.

The method you choose must be defined in the configuration key `BM_ARCHIVE_METHOD`. You can put here a list of all the different methods you want to use. Take care to put every configuration key needed by all the methods you choose. Note that you can also choose none of the proposed methods, if you don't want to build archives with this configuration file, then just put `none`.

A couple of other configuration keys may be needed depending on the method you choose.

Example:

```
export BM_ARCHIVE_METHOD="tarball-incremental mysql"
```

2.2.1 Tarballs

Description

Method name: `tarball`, configuration key prefix: `BM_TARBALL`.

If all you want to do is to handle a couple of tarballs of your file system, you can use this method. This method takes a list of directories and builds the corresponding tarballs. This method is the default one, this is the easiest to use, it just builds tarballs as you could do with your own tar script. Its main drawback is to eat a lot of disk space: archives can be big from a day to another, even if there are no changes in their content. See the `tarball-incremental` method if you want to optimize archives' size.

When building full backups (when not building incremental ones), Backup Manager will append the keyword "master" to the name of the archive. This is very useful when using the `tarball-incremental` method for seeing where the full backups are quickly.

A couple of options are available: the name format of the archive, the compression type (gzip, zip, bzip2, none) and the facility to dereference symlinks when building the tarball.

BM_TARBALL_NAMEFORMAT

This configuration key defines how to perform the naming of the archive. Two values are possible:

- `long`: the name will be made with the absolute path of the directory (eg: `var-log-apache` for `/var/log/apache`).
- `short`: the name will just contain the directory (eg: `apache` for `/var/log/apache`).

Suggested value: `long`.

BM_TARBALL_FILETYPE

Type: enum(tar, tar.gz, tar.bz2, tar.xz, tar.lzma, zip, dar), default: `tar.gz`.

Basically, this configuration key defines the filetype of the resulting archive. In a way, it defines which compressor to use (zip, gzip, dar or bzip2). Here are the supported values: `tar`, `tar.gz`, `tar.bz2`, `zip` and `dar`. Note that depending on the filetype you choose, you will have to make sure you have the corresponding compressor installed.

For the best compression rate, choose `tar.bz2` or `tar.xz`.

Since version 0.7.1, Backup Manager supports *dar* archives. This archiver provides some interesting features like the archive slicing.

Since version 0.7.5, Backup Manager supports *lzma* archives.

Make sure to statisfy dependencies according to the filetype you choose:

- tar.bz2 : needs “bzip2”.
- tar.xz : needs “xz”.
- tar.lzma : needs “lzma”.
- dar : needs “dar”.
- zip : needs “zip”.

BM_TARBALL_SLICESIZE

Type: string

If you want to make sure your archives won’t exceed a given size (for instance 2 GB) you can use that configuration variable, but only if you are using the `dar` `BM_TARBALL_FILETYPE`. Indeed this feature is only supported by dar.

If you want to limit your archives size to 1 giga byte, use such a statement:

```
BM_TARBALL_SLICESIZE="1000M"
```

Refer to the dar manpage for details about slices.

BM_TARBALL_EXTRA_OPTIONS

Type: string

If you want to provide extra options to “tar” or “dar” you may do so here. Leave blank unless you know what you are doing.

Example: to enable verbosity with tar (which would appeard in the logfiles), use this:

```
BM_TARBALL_EXTRA_OPTIONS="-v"
```

BM_TARBALL_DUMPSYMLINKS

Type: boolean, default: `true`.

It is possible, when generating the tarball (or the zip file) to dereference the symlinks. If you enable this feature, every symbolic link in the file system will be replaced in the archive by the file it points to. Use this feature with care, it can quickly lead to huge archives, or even worse: if you have a circular symlink somewhere, this will lead to an infinite archive!

In most of the cases, you should not use this feature.

BM_TARBALL_DIRECTORIES

Type: space-separated list, default: `null`.

Since version 0.7.3, this variable is replaced by the array BM_TARBALL_TARGETS[], it's still supported for backward compatibility though. You can use this variable for defining the locations to backup, but you must not use this variable if one or more of the paths you want to archive contain a space.

If you want to backup some targets that have spaces in their name (eg "Program Files"), you must not use this variable, but the array BM_TARBALL_TARGETS[] instead.

Example:

```
export BM_TARBALL_DIRECTORIES="/etc /home /var/log/apache"
```

BM_TARBALL_TARGETS

Type: array, default: `"/etc", "/boot"`.

This variable holds every place you want to backup. This is the recommanded variable to use for defining your backup targets (`BM_TARBALL_DIRECTORIES` is deprecated since version 0.7.3).

You can safely put items that contain spaces (eg: "Program Files") whereas you can't with `BM_TARBALL_DIRECTORIES`.

You can also put Bash patterns in BM_TARBALL_TARGETS[], it will be expanded at runtime to find the resulting targets. For instance : BM_TARBALL_TARGETS[0]="/home/*" will lead to backup every home's sub-directory.

Example

```
BM_TARBALL_TARGETS[0]="/etc"
BM_TARBALL_TARGETS[1]="/home/*"
BM_TARBALL_TARGETS[2]="/boot"
BM_TARBALL_TARGETS[3]="/mnt/win/Program Files"
```

BM_TARBALL_BLACKLIST

Type: space-separated list, default: `"/proc /dev /sys /tmp"`.

It can be very useful to prevent some locations of your filesytem from being included in the archives. This is really useful when you use wildcards in BM_TARBALL_DIRECTORIES. Indeed, you may want to backup every top-level directory of your filesystem (`/*`) but without volatile locations like `/tmp`, `/dev` and `/proc`.

You can also use this variable for excluding every files of a given extension, like for instance mp3 or mpg files.

Example:

```
export BM_TARBALL_BLACKLIST="/tmp /dev /proc *.mp3 *.mpg"
```

BM_TARBALL_OVER_SSH

Type: boolean, default: `false`.

Dependency: BM_UPLOAD_SSH

If you want to archive some remote locations from a server where Backup Manager is insalled, you can choose to build archives over SSH. This is useful if you don't want to install Backup Manager every where and setup some upload methods from all thoses servers to a central data storage server. This way, Backup Manager will build some archives directly over SSH and will store the resulting tarballs locally, as if it was built like any other archive. The resulting archive will be prefixed with the remote hostname instead of `BM_ARCHIVE_PREFIX`.

This feature requires that the following variables are set in the BM_UPLOAD_SSH section:

- `BM_UPLOAD_SSH_USER`: the user to use for connecting to the remote server. Note that this user will run tar remotely, so take care to archive something this user can read!
- `BM_UPLOAD_SSH_KEY`: as usal, the path to the private key to use for establishing the connection.
- `BM_UPLOAD_SSH_HOSTS`: A list of hosts where to run the tarball builds.

If you enable this feature, note that the resulting configuration file will have the following restrictions:

- Remote tarball build only works with the `tarball` method, it will silently behaves the same with `tarball-incremental`.
- You cannot use the remote build and the local one in the same configuration file. If you want to do both, use two configuration files.

Example: You have three hosts: host01, host02 and host03. You want to set up host01 as a data storage server, it has a big /var/archives partition. You want to archive "/etc", "/home" and "/var/log" on box02 and box03 and store the archives on host01.

```
[...]
export BM_ARCHIVE_METHOD="tarball"

export BM_TARBALL_OVER_SSH="true"
export BM_TARBALL_FILETYPE="tar.bz2"
export BM_TARBALL_DIRECTORIES="/etc /home /var/log"

export BM_UPLOAD_SSH_USER="bamuser"
export BM_UPLOAD_SSH_KEY="/home/bamuser/.ssh/id_dsa"
export BM_UPLOAD_SSH_HOSTS="box02 box03"
```

Of course, for this to work correctly, `'bamuser'` should be a valid user on box02 and box03; it must be allowed to connect to them with SSH key autentication and has to be able to read those directories.

2.2.2 Incremental tarballs

Description

Method name: `tarball-incremental`, configuration key prefix: `BM_TARBALLINC`.

If you want to handle tarballs without wasting disk space, you should use this method. The concept of this method is simple: You choose a frequency when a full backup is made (exactly like the one made by the tarball mehod). All the days between two full backups, archives contain only the files that have changed from the previous archive.

For instance, let's say you want to backup /home with this method. Your /home directory is composed by two sub-directories: /home/foo and /home/bar. You choose a weekly frequency and say that monday will be the "fullbackup" day. Obviously, you will have a full tarball of /home on monday. Then, if a file changed inside /home/foo and if /home/bar remains unchanged, tuesday's archive will only contain the modified files of /home/foo. Using this method will save a lot of disk space.

To build incremental tarballs, Backup Manager uses tar's switch `--listed-incremental`. This will create a file for each target which will contain some statistics used by tar to figure out if a file should be backed up or not. When Backup Manager is run for the first time, this file doesn't exist, so the first tarballs made are always master backups. If the *incremental list* files get removed, the next backups won't be incremental.

Since version 0.7.3, it's possible to see at the first glance if a backup is a master or an incremental one: master backup have the keyword `master` appended to the date. When purging the repository, the master backups are not removed as the incremental ones. Backup Manager always keep a master backup that is older than incremental archives.

This method uses all the tarball's configuration keys and adds two more. One to define the kind of frequency, the other to choose on which day the full backups should be done.

`BM_TARBALLINC_MASTERDATETYPE`

Type: enum(weekly, monthly), default: `weekly`.

This is the type of frequency you want to use. If you choose `weekly`, you'll have to choose a day number between 0 and 6 for the BM_TARBALLINC_MASTERDATEVALUE configuration key, if you choose `monthly`, the day number will be between 1 and 31.

`BM_TARBALLINC_MASTERDATEVALUE`

Type: integer, default: `1`.

The number of the day when making full backups. Note that its meaning directly depends on the `BM_TARBALLINC_MASTERDATETYPE`. For instance, 1 means *"monday"* if you choose a weekly frequency, but it means *"the first day of the month"* if you choose a monthly frequency.

2.2.3 MySQL databases

Description

Method name: `mysql`, configuration keys prefix: `BM_MYSQL`.

This method provides a way to archive MySQL databases, the archives are made with mysqldump (SQL text files) and can be compressed.

In versions prior to 0.7.6, Backup Manager used to pass the MySQL client's password through the command line. As explained by the MySQL manual, that's a security issue as the password is then readable for a short time in the /proc directory (or using the ps command).

To close that vulnerability, the MySQL client password is not passed through the command line anymore, it is written in a configuration file located in the home directory of the user running Backup Manager : `~/.my.cnf`.

If that file doesn't exist at runtime, Backup Manager will create it and will then write the password provided in `BM_MYSQL_ADMINPASS` inside.

BM_MYSQL_DATABASES

Type: space-separated list, default: `__ALL__`.

This is the list of databases you want to archive. You can put the keyword `__ALL__` if you like to backup every database without having to list them.

Example:

```
export BM_MYSQL_DATABASES="mysql mybase wordpress dotclear phpbb2"
```

BM_MYSQL_SAFEDUMPS

Type: boolean, default: `true`.

The best way to produce MySQL dumps is done by using mysqldump's `--opt` switch. This makes the dump directly usable with mysql (adds the drop table statements), locks tables during the dump generation and other cool things (see `mysqldump`). This is recommended for full-clean-safe backups, but needs a privileged user (for the lock permissions).

Example:

```
export BM_MYSQL_SAFEDUMPS="true"
```

BM_MYSQL_ADMINLOGIN

Type: string, default: `root`.

The MySQL login you want to use for connecting to the database. Make sure this login can read all the databases you've set in `BM_MYSQL_DATABASES`.

Example:

```
export BM_MYSQL_ADMINLOGIN="root"
```

BM_MYSQL_ADMINPASS

Type: string, default: `undefined`.

The MySQL client password.

If you have already made your own ~/.my.cnf configuration file, you don't have to set that variable.

If you don't know what is the `~/.my.cnf` configuration file, set the password, then Backup Manager will take care of creating the MySQL client configuration file.

Example:

```
export BM_MYSQL_ADMINPASS="MySecretPass"
```

BM_MYSQL_HOST

Type: string, default: `localhost`.

The database host where the databases are.

Example:

```
export BM_MYSQL_HOST="localhost"
```

BM_MYSQL_PORT

Type: string, default: `3306`.

The port on `BM_MYSQL_HOST` where the mysql server is listening.

Example:

```
export BM_MYSQL_PORT="3306"
```

BM_MYSQL_FILETYPE

Type: enum(gzip, bzip2), default: `bzip2`.

The archive is made with mysqldump which renders SQL lines; the resulting text file can be compressed. If you want to compress the file, choose the compressor you want. Leave it blank if you want pure SQL files.

Example:

```
export BM_MYSQL_FILETYPE="bzip2"
```

2.2.4 Subversion repositories

Description

You can archive Subversion repositories with this method. The archive will be made with `svnadmin` and will contain XML data (text files). Like the mysql method, you can choose to compress it.

BM_SVN_REPOSITORIES

Type: space-separated list

This is the list of absolute paths to the SVN repositories to archive.

Example:

```
export BM_SVN_REPOSITORIES="/srv/svnroot/repo1 /srv/svnroot/repo2"
```

BM_SVN_COMPRESSWITH

Type: enum(gzip, bzip2), default: `bzip2`.

If you want to compress the resulting XML files, choose a compressor here. Leave this blank if you don't want any compression.

Example:

```
export BM_SVN_COMPRESSWITH="gzip"
```

2.2.5 Generic methods

Description

Even if most of the common needs are covered by the existing methods, there is always a case uncovered. Backup Manager provides a way for backing up anything, and can be used in such circumstances.

This method is called `pipe`, it is more complex to use but can virtually backup anything. The concept is simple, a pipe method is defined by the following items:

- A name (for naming the archive)
- A command (that produces content on stdout)
- A file type (txt, sql, dump, ...)
- A compressor (gzip, bzip2)

Those configuration keys are arrays, so you can implement as many pipe methods as you like.

For each pipe method defined, Backup Manager will launch the command given and redirect the content sent to stdout by this command to a file named with the name of the method and its filetype. Then, if the method uses a compressor, the file will be compressed.

Example

Example for archiving a remote MySQL database through SSH:

```
BM_PIPE_COMMAND[0]="ssh host -c \"mysqldump -ufoo -pbar base\""
BM_PIPE_NAME[0]="base"
BM_PIPE_FILETYPE[0]="sql"
BM_PIPE_COMPRESS[0]="gzip"
```

Imagine you have a second pipe method to implement, for instance building a tarball trough SSH:

```
BM_PIPE_COMMAND[1]="ssh host -c \"tar -c -z /home/user\""
BM_PIPE_NAME[1]="host.home.user"
BM_PIPE_FILETYPE[1]="tar.gz"
BM_PIPE_COMPRESS[1]=""
```

Note that we have incremented the array's index.

2.3 Upload Methods

2.3.1 Description

One of the most important thing to do when backing up file systems is to store the archives on different places. The more different physical spaces you have, the better. Backup Manager provides a way for achieving this goal : the upload methods.

There are different upload methods, each of them behaves differently and provides particular features. In Backup Manager 0.7.7 you can use FTP, SSH, RSYNC or Amazon S3 uploads.

In the same manner as for backup methods, you can choose to use as many upload methods as you like. If you don't want to use this feature at all, just put the keyword `none` in the configuration `BM_UPLOAD_METHOD`.

Note that the FTP, SSH and S3 methods are dedicated to upload archives, using those method depends on the use of at least one backup method.

On the opposite, the RSYNC method uploads a directory to remote locations, this directory can be your repository or whatever other location of your file sytem.

2.3.2 Global configuration keys

The following configuration keys are global in the upload section:

BM_UPLOAD_HOSTS

Type: space-separated list

Each of the hosts defined in that list is used by all the upload methods when establishing connections. For instance if you want to perform SSH uploads of your archives and RSYNC upload of a location to the same host, put it in this list.

Example:

```
export BM_UPLOAD_HOSTS="mirror1.lan.mysite.net mirror2.lan.mysite.net"
```

BM_UPLOAD_DESTINATION

Type: string

This is the absolute path of the directory in the remote hosts where to put the files uploaded.

If you have installed installed Backup Manager on the remote host, a good idea is to choose a sub-directory of the repository. Then, during the remote host purge phase, your uploads will be cleaned at the same time.

You can also define a destination dedicated to your host: `BM_UPLOAD_DESTINATION="/var/archives/$HOSTNAME"`

Example:

Let's say you want that all your uploads are performed on the host mirror2.lan.mysite.net, in the sub-directory /var/archives/uploads

```
export BM_UPLOAD_HOSTS="mirror2.lan.mysite.net"
export BM_UPLOAD_DESTINATION="/var/archives/uploads"
```

2.3.3 SSH uploads

Description

Method name: `ssh`, goal: upload archives to remote hosts over SSH. This method depends on a backup method.

If you want to upload your archives on remote locations, you can use the SSH method. This method is good if you like to use a secure tunnel between the two points of the upload.

The call to `scp` will be done with the identity of the user `BM_UPLOAD_SSH_USER`, thus, you have to make sure this user can have access to the repository (take care to the secure mode).

BM_UPLOAD_SSH_USER

Type: string

This is the user to use for performing the ssh connection. Make sure this user can access repository.

Example:

```
export BM_UPLOAD_SSH_USER="bmngr"
```

BM_UPLOAD_SSH_KEY

Type: string

This is the path to the private key of the user BM_UPLOAD_SSH_USER.

Example:

```
export BM_UPLOAD_SSH_KEY="/home/bmngr/.ssh/id_dsa"
```

BM_UPLOAD_SSH_PORT

Type: integer

You may want to connect to remote hosts with a specific port. Use this configuration key then.

Example:

```
export BM_UPLOAD_SSH_PORT="1352"
```

BM_UPLOAD_SSH_HOSTS

Type: space-separated list

Put here the list of hosts to use for SSH-only uploads. Note that if you put some hosts in `BM_UPLOAD_HOSTS`, they will be used as well.

Example:

```
export BM_UPLOAD_SSH_HOSTS="mirror3.lan.mysite.net"
```

BM_UPLOAD_SSH_PURGE

Type: boolean

If you set this boolean to "true", the remote archives will be purged before the new ones are uploaded. The purging rules are the same as the ones Backup Manager uses for local purging. If `BM_UPLOAD_SSH_TTL` is defined, this time to live will be used, else `BM_ARCHIVE_TTL` will be used.

Example:

```
export BM_UPLOAD_SSH_PURGE="true"
export BM_UPLOAD_SSH_TTL="10"
```

BM_UPLOAD_SSH_DESTINATION

Type: string

Put here the destination for SSH-only uploads, this key overrides `BM_UPLOAD_DESTINATION`.

Example:

```
export BM_UPLOAD_SSH_DESTINATION="/var/archives/scp-uploads"
```

2.3.4 Encrypted SSH uploads

Description

Method name: `ssh-gpg`, goal: encrypt arcives using public key encryption and upload the result to untrusted remote hosts over SSH. This method depends on a backup method.

The upload using SSH can also be combined with public key encryption provided by `gpg`. The archives will be encrypted using a public key prior to sending them over the network, so on the remote server your files are protected from inspection.

This method can be used to protect your data from inspection on untrusted remote servers. However, since the encrypted files are not signed, this does not protect you from archive manipulation. So the *md5* hases are still needed.

This method uses all of the configuartion keys of the *ssh* method. One additional key is required.

BM_UPLOAD_SSH_GPG_RECIPIENT

Type: string

This parameter sets the recipient for which the archive is encrypted. A valid specification is a short or long key id, or a descriptive name, as explained in the `gpg` man page. The public key for this identity must be in the key ring of the user running `gpg`, which is the same as specified by `BM_UPLOAD_SSH_USER`. To test this run the command `gpg --list-keys ID` as that user, where `ID` is the ID as you give it to this parameter. If `gpg` displays exactly one key, then you are fine. Refer to the `gpg` man page for further details.

Example:

```
export BM_UPLOAD_SSH_GPG_RECIPIENT="email@address.com"
export BM_UPLOAD_SSH_GPG_RECIPIENT="ECE009856"
```

2.3.5 FTP uploads

Description

If security does not matter much on your lan (between the two points of the upload) you can choose to use the FTP method. One of the main pros of this method is that it can perform purging independently. You can safely use this method for uploading files to a host where you just have an FTP account.

BM_UPLOAD_FTP_SECURE

Type: boolean, default: false.

If this variable is set to true, all FTP transfers will be done over SSL.

Example:

```
export BM_UPLOAD_FTP_SECURE="true"
```

BM_UPLOAD_FTP_PASSIVE

Type: boolean, default: true.

If this variable is set to true, FTP transfers will be performed in passive mode, which is mandatory in NATed/firewalled environments.

Example:

```
export BM_UPLOAD_FTP_PASSIVE="true"
```

BM_UPLOAD_FTP_TTL

Type: integer, default: $BM_ARCHIVE_TTL

Using different *time to live* values for local and remote archives can be useful in certain situations. For instance, it's possible to install Backup Manager locally, make it build archives, upload them to a remote FTP host and then purge them locally (but not on the remote host). Doing this is possible with setting a null value to the local TTL (BM_ARCHIVE_TTL) and a non-null value to BM_UPLOAD_FTP_TTL.

Example:

```
# in your main conffile -- /etc/backup-manager.conf
export BM_ARCHIVE_TTL="0"
export BM_UPLOAD_FTP_TTL="5"
export BM_POST_BACKUP_COMMAND="/usr/sbin/backup-manager --purge"

# in your cron job:
/usr/sbin/backup-manager
/usr/sbin/backup-manager --purge

(Don't put the post-command in the main conffile or you'll face an infinite loop.)
```

BM_UPLOAD_FTP_USER

Type: string.

Put here the FTP user to use for opening the connections.

Example:

```
export BM_UPLOAD_FTP_USER="bmngr"
```

BM_UPLOAD_FTP_PASSWORD

Type: string.

Put here the password to use for authenticating the FTP session,(in plain text).

Example:

```
export BM_UPLOAD_FTP_PASSWORD="secret"
```

BM_UPLOAD_FTP_HOSTS

Type: space-separated list

Put here the list of hosts to use for FTP-only uploads. Note that if you put some hosts in `BM_UPLOAD_HOSTS`, they will be used as well.

Example:

```
export BM_UPLOAD_FTP_HOSTS="mirror4.lan.mysite.net"
```

BM_UPLOAD_FTP_DESTINATION

Type: string

Put here the destination for FTP-only uploads, this key overrides `BM_UPLOAD_DESTINATION`.

Example:

```
export BM_UPLOAD_FTP_DESTINATION="/var/archives/ftp-uploads"
```

BM_UPLOAD_FTP_PURGE

Type: boolean, default: `true`

You can choose to purge deprecated archives before uploading new ones. This purge is done over FTP and uses the configuration key `BM_ARCHIVE_TTL` in the same manner as the local purge behaves (the FTP purge is not recursive though).

Example:

```
export BM_UPLOAD_FTP_PURGE="true"
```

2.3.6 Amazon S3 uploads

Description

Amazon's new Simple Storage Service (S3) is an Internet "web service" that permits you to store unlimited blocks of data on their replicated and managed systems. See http://aws.amazon.com for more information. Registration is free and the rates are quite reasonable.

Using the S3 upload method will permit your archives to be stored on Amazon's S3 service. You must allocate a "bucket" to the exclusive use of Backup Manager. Each of your created archives will be uploaded to S3 and stored within this bucket in a key name that matches the name of the archive.

As with the other backup methods Backup Manager does not assist you in restoring files from archives. You must retrieve archives from S3 using other mechanisms such as the S3Shell provided as an examle command line utility by Amazon.

Note that when using this upload method, the `BM_UPLOAD_HOSTS` variable is ignored as the only valid host for S3 uploads in `s3.amazon.com`.

BM_UPLOAD_S3_DESTINATION

Type: string.

This option is required for the S3 upload method. This specifies the bucket used to store backup data. If the bucket does not exist it will be created as a private bucket. This key overrides `BM_UPLOAD_DESTINATION`. Note that Amazon requires that bucket names be globally unique. Be creative picking one.

Example:

```
export BM_UPLOAD_S3_DESTINATION="my_backup_bucket"
```

BM_UPLOAD_S3_ACCESS_KEY

Type: string.

This option is required for the S3 upload method. After you have registered Amazon will provide you an access key. You must use this key to access your storage on S3.

Example:

```
export BM_UPLOAD_S3_ACCESS_KEY="a9sabkz0342dasv"
```

BM_UPLOAD_S3_SECRET_KEY

Type: string.

This option is required for the S3 upload method. After you have registered Amazon will provide you a secret key. You must use this key to write to your storage on S3.

Example:

```
export BM_UPLOAD_S3_SECRET_KEY="lkj2341askj123sa"
```

BM_UPLOAD_S3_PURGE

Type: boolean, default: `true`

You can choose to purge deprecated archives before uploading new ones. This purge is done over S3 and uses the configuration key `BM_ARCHIVE_TTL` in the same manner as the local purge behaves (the S3 purge is not recursive though).

Example:

```
export BM_UPLOAD_S3_PURGE="true"
```

BM_UPLOAD_S3_TTL

Type: integer, default: $BM_ARCHIVE_TTL

Using different *time to live* values for local and remote archives can be useful in certain situations. For instance, it's possible to install Backup Manager locally, make it build archives, upload them to S3 and then purge them locally (but not on the remote host). Doing this is possible with setting a null value to the local TTL (BM_ARCHIVE_TTL) and a non-null value to BM_UPLOAD_S3_TTL.

2.3.7 RSYNC uploads

Description

You may want to upload some parts of your file system to some remote hosts. In these cases, archives are not needed, you just want to synchronize some directories to remote places. This is where the RSYNC upload method is useful.

RSYNC uploads need a SSH user/key pair to behave correctly, thus there is a dependency against the keys `BM_UPLOAD_SSH_USER` and `BM_UPLOAD_SSH_KEY`.

BM_UPLOAD_RSYNC_DIRECTORIES

Type: space-separated list

Put here the list of local directories you want to upload with rsync.

Example:

```
export BM_UPLOAD_RSYNC_DIRECTORIES="/data/photos /data/videos /data/mp3"
```

BM_UPLOAD_RSYNC_HOSTS

Type: space-separated list

Put here the list of hosts to use for RSYNC-only uploads. Note that if you put some hosts in `BM_UPLOAD_HOSTS`, they will be used as well.

Example:

```
export BM_UPLOAD_RSYNC_HOSTS="mirror5.lan.mysite.net"
```

BM_UPLOAD_RSYNC_DESTINATION

Type: string

Put here the destination for RSYNC-only uploads, this key overrides `BM_UPLOAD_DESTINATION`.

Example:

```
export BM_UPLOAD_RSYNC_DESTINATION="/var/archives/rsync-snapshots"
```

BM_UPLOAD_RSYNC_DUMPSYMLINKS

Type: boolean, default: `false`.

You can choose to dereference files pointed by symlinks in your RSYNC snapshots. This feature should be used with care.

Example:

```
export BM_UPLOAD_RSYNC_DUMPSYMLINKS="false"
```

2.4 Exports

Another way of storing your archives to a safe place is to use external media.

In version 0.7.7, only CDs and DVDs are supported as external media, so we will discuss in this section only the `BM_BURNING` features. Other exports are expected to come in next versions though.

2.4.1 Burning CDR/DVD media

In the version 0.7.7, Backup Manager supports four different kinds of media: CDR, CDRW and DVD+R(W) and DVD-R(W).

BM_BURNING_METHOD

Set the key `BM_BURNING_METHOD` to the method corresponding to the media you want to burn:

- CDR
- CDRW
- DVD

- DVD-RW

In *non-interactive mode* (when backup-manager is not lauchned from a terminal), any of these methods will try to put the whole archive repository in the media, if it does not fit in the media, it will try to put only the archives built on the day, if that's not possible, nothing will be burnt.

In *interactive mode* (when backup-manager is launched from a terminal), the whole repository will be burnt into as many media as needed. When a medium is satured with archives, backup-manager will pause the process asking the user to put a new media inside.

The `CDRW` and `DVD-RW` methods will first blank the media, so you can safely use these methods if you want to use the same medium several times.

The `CDR` and `DVD` medthods won't blank the medium first (DVD+RW media doesn't need blanking, it's possible to re-burn data on-the-fly over such media)..

DVD media are handled by the tool `dvd+rw-tools`, problems can occur in CRON environment with `dvd+rw-tools` versions prior to `6.1`, make sure to have `6.1` or later if you want to burn DVD media with Backup Manager.

As usual, you can put `none` in order to disable the burning process.

All those burning methods share the same configuration keys, so it's easy to switch from a medium to another.

BM_BURNING_DEVICE

Type: string, default: `/dev/cdrom`.

This is mandatory for using the burning feature, it's the device to use for mounting the media. It's needed by backup manager for performing the MD5 checks and for other needs.

Example:

```
export BM_BURNING_DEVICE="/dev/cdrom"
```

BM_BURNING_DEVFORCED

Type: string

Backup Manager uses `cdrecord` for burning CDs. If when you run `cdrecord -scanbus` you don't see your burning device, that means you will have to force the device in ATA mode. To tell Backup Manager to do so, just put here the path to your device, and a switch will be appended to the cdrecord commandline like the following : `cdrecrord ... dev=$BM_BURNING_DEVFORCED ...`.

Leave this configuration key blank if you see your device with `cdrecord -scanbus`, in this case, Backup Manager will use the default cdrecord device for burning CDR media.

Example:

```
export BM_BURNING_DEVFORCED="/dev/cdrom"
```

BM_BURNING_ISO_FLAGS

Type: string, default: "-R -J"

Media burned with Backup Manager will be made using a Joliet disc image. The flags defined in that variable will be appended to the mkisofs command lines in order to specify wich media image to use.

The default value "-R -J" produces a Joliet image, if you want to make non-Joliet disc images, you can change these flags. Refer to the manpage of mkisofs for details about possible disc images.

Don't change that variable if you don't know what you're doing.

Example:

```
export BM_BURNING_ISO_FLAGS="-R -J"
```

BM_BURNING_MAXSIZE

Type: integer, default: `700`.

This is where you define the maximum size (in megabytes) of the media you will put in the device. Here is the list of the common sizes:

- CDR/CDRW: 650, 700, 800
- DVD: 4700

When Backup Manager looks in the repository for burning data, it will try to put the whole archive repository in the media. If the summarized size of the repository does not fit in `BM_BURNING_MAXSIZE`, Backup Manager will then try to put only the archives of the day.

Example for a CD burner

```
export BM_BURNING_METHOD="CDRW"
export BM_BURNING_MAXSIZE="700"
```

Example for a DVD burner:

```
export BM_BURNING_METHOD="DVD"
export BM_BURNING_MAXSIZE="4700"
```

BM_BURNING_CHKMD5

Type: boolean, default: `true`.

If this boolean is set to a true value, every MD5 sum will be checked when the media is burned in order to make sure everything is ok.

Note that you can choose to perform this checkup with the command switch `--md5check`.

Example:

```
exports BM_BURNING_CHKMD5="true"
```

2.5 Advanced features

A couple of advanced features are provided, they will be covered in this section.

2.5.1 BM_TEMP_DIR

Type: string, default: `/tmp/backup-manager`.

This is the temporary directory where temporary files are created by Backup Manager.

Example:

```
export BM_ARCHIVE_CHMOD="/tmp/backup-manager"
```

2.5.2 Logging to syslog

If you want to log Backup Manager actions to syslog, you can enable the internal logger, this is done with the configuration key `BM_LOGGER`. You are also able to choose which syslog facility to use thanks to the key `BM_LOGGER_FACILITY`.

BM_LOGGER

Type: boolean, default: `true`.

If this boolean is set to true, Backup Manager will log everything to syslog.

Example:

```
exports BM_LOGGER="true"
```

BM_LOGGER_FACILITY

Type: string, *default:* `user`.

You can specify here a syslog facility to use, this can be useful if you like to filter messages from Backup Manager to a special syslog file.

Example:

```
exports BM_LOGGER_FACILITY="cron"
```

2.5.3 Writing external hooks

You have the availability to write your own hooks if you want to automate some special beaviours within the Backup Manager process. You may like to mount over NFS your archive repository *before* the backup session and unmount it after, or you may like to launch your own uploader script when the backup session is finished.

In order to let you implement any solution you like, Backup Manager provides two different hooks: the *pre-command* and *post-command* hooks.

BM_PRE_BACKUP_COMMAND

Type: string

Put here the path to a program (or a shell command) to launch before the backup session. If the command fails (exits with non zero value, or prints the keyword `false` on stdout) the backup session will stop. If the pre-command succeeds, the process can follow.

Example with a basic shell command:

```
export BM_PRE_BACKUP_COMMAND="mount -t nfs mirror.lan.net:/exports/backups /var/archives"
```

Example with a custom script:

```
export BM_PRE_BACKUP_COMMAND="/usr/local/bin/backup-prepare.pl $TODAY"
```

BM_POST_BACKUP_COMMAND

Type: string

Put here the path to a program (or a shell command) to launch after the backup session. If the command fails (exits with non zero value, or prints the keyword `false` on stdout) Backup Manager will exit with an error code (and will log to syslog the post-command failure if the logger is enabled).

Example with a basic shell command:

```
export BM_POST_BACKUP_COMMAND="umount /var/archives"
```

Example with a custom script:

```
export BM_POST_BACKUP_COMMAND="/usr/local/bin/backup-cleanup.pl $TODAY"
```

Chapter 3

Using Backup Manager

Now that you know in details how to write your configuration files, let's see how to use Backup Manager.

3.1 Command line

3.1.1 Restrictions

In version 0.7.7, Backup Manager can only be used by `root`, as it has be designed as a systemwide tool.

```
$ backup-manager
backup-manager must be run as root.
```

If you want to launch it from the command line, you first have to use the `root` account.

```
$ su
Password:
# backup-manager -h
/usr/sbin/backup-manager [options]

Output:
--help|-h           : Print this short help message.
--verbose|-v        : Print what happens on STDOUT.
--no-warnings       : Disable warnings.

Single actions:
--upload|-u         : Just upload the files of the day.
--burn|-b           : Just burn the files of the day.
--md5check|-m       : Just test the md5 sums.
--purge|-p          : Just purge old archives.

Behaviour:
--conffile|-c file  : Choose an alternate config file.
--force|-f          : Force overwrite of existing archives.

Unwanted actions:
--no-upload         : Disable the upload process.
--no-burn           : Disable the burning process.
--no-purge          : Disable the purge process.
ouranos:/home/sukria#
```

As you can see in the example above, using the `-h` switch (or `--help`) gives a short help message and prints all supported command switches. We will cover in this section each of them.

3.1.2 Options

The following switches can be used for altering Backup Manager's behaviour.

`--version`

Prints on stdout the Backup Manager version installed on the system and exit.

Example:

```
# backup-manager --version
Backup Manager 0.6
```

`--verbose` or `-v`

Using this switch will enabled the verbose mode. All actions are reported on stdout.

Example:

```
# backup-manager -v
Getting lock for backup-manager 10605 with /etc/backup-manager.conf: ok
Cleaning /var/archives
Entering directory /var/archives/lost+found.
[...]
```

`--no-warnings`

When a non-critical problem occurs (an error occured but the backup process can follow) Backup Manager will print a warning message (and will log it if the logger is enabled). If you don't want to see warning messages, you can append this switch on the command line.

`--conffile` or `-c`

Backup Manager relies on configuration files, by default, the file `/etc/backup-manager.conf` is used but you can choose to run it with a different one. This is done by using the following syntax :

```
# backup-manager -c <FILE>
```

Note that Backup Manager is designed to work properly when launched in parallel mode with different configuration files, but it will refuse to run twice at the same time with the same configuration file. You can then safely do something like that:

```
# backup-manager -c /etc/backup-manager/backup-nfs.conf &
# backup-manager -c /etc/backup-manager/backup-homedirs.conf &
# backup-manager -c /etc/backup-manager/backup-rsync-filer.conf
```

`--force`

When building an archive, Backup Manager looks if the archive already exists in the repository, if so, a warning is sent saying that the archive exists. If you want to bypass this warning and overwrite archives, use this switch.

`--upload` or `-u`

If you have made a configuration file that enables the uploading system, you can ask Backup Manager to perform the uploading session instead of the whole process with this switch.

`--burn` or `-b` [<DATE>]

If you have made a configuration file that enables the burning system, you can ask Backup Manager to perform the burning session instead of the whole process with this switch.

You can also ask Backup Manager to burn only archives of a given date with providing a timestamp after the `--burn` switch.

Example:

Burning all the archives made on March, 12nd 2006:

```
# backup-manager --bnurn 20060312
```

`--md5check` or `-m`

If you have made a configuration file that enables the MD5 checks on burnt media, you can ask Backup Manager to perform the MD5 checks instead of the whole process with this switch.

--purge or -p

This switch will as Backup Manager to just perform the archive repository purge: removing any depreacted archives (according to `BM_ARCHIVE_TTL`.

--no-upload or -p

Use this switch if you have a configuration file that enables the uploading system and want to run Backup Manager without it.

--no-burn

Use this switch if you have a configuration file that enables the burning system and want to run Backup Manager without it.

--no-purge or -p

Use this switch if you want to disable the purging phase. This can be useful if you like to implement another kind of purging system, with a post-command hook for instance.

3.2 CRON integration

There is a global idea behind Backup Manager's design: "*You won't do it if you have to think about it*". This is specifically true for backup concerns and it is strongly adviced to automate your backup process with a tasks scheduler like CRON.

Setting up a Backup Manager job in cron is pretty easy, you just have to write a shell script under the appropriate CRON sub-directory that will call backup-manager. The best sub-directory to choose is `/etc/cron.daily` as Backup Manager handles daily archives.

Here is an example of a CRON script:

```
cat > /etc/cron.daily/backup-manager
#!/bin/sh

/usr/sbin/backup-manager
```

If you want to be notified by mail if a problem occurs during the backup session, just make sure you receive mails coming from CRON. When the verbose mode is off, only warnings and errors are printed on stdout, so you will receive a mail from the Backup Manager CRON job only in case of unexpected effects.

On the other hand, if you like to receive daily mails from the job, even if everything went well, just append the –verbose switch like that :

```
cat > /etc/cron.daily/backup-manager
#!/bin/sh

/usr/sbin/backup-manager --verbose
```