Emborg Documentation

Release 1.38

Ken Kundert

Nov 05, 2023

CONTENTS

Version: 1.38
Released: 2023-11-04
Please report all bugs and suggestions on GitHub.

CHAPTER

ONE

WHAT IS EMBORG?

Emborg is a simple command line utility to orchestrate backups. It is built as a front-end to Borg, a powerful and fast de-duplicating backup program. With *Emborg*, you specify all the details about your backups once in advance, and then use a very simple command line interface for your day-to-day activities.

Use of *Emborg* does not preclude the use of *Borg* directly on the same repository. The philosophy of *Emborg* is to provide commands that you would use often and in an interactive manner with the expectation that you would use *Borg* directly for more unusual or esoteric situations.

CHAPTER

TWO

WHY EMBORG?

There are alternatives to *Emborg* such as BorgMatic and Vorta, both of which are also front-ends to *BorgBackup*. *BorgMatic* has a command line interface like *Emborg* while *Vorta* is GUI-based. *Emborg* distinguishes itself by providing a command line interface that is very efficient for common tasks, such as creating archives (backups), restoring files or directories, or comparing existing files to those in an archive. Also, *Emborg* naturally supports multiple destination repositories. This feature can be used to simultaneously backup to a local repository, which provides rapid restores, and an off-site repository, which provides increased safety in case of a local disaster.

CHAPTER

THREE

WHY BORG?

Well, everyone needs to backup their files. So perhaps the questions should be: why not *Duplicity*? Duplicity has been the standard way to do backups on Unix systems for many years.

Duplicity provides full and incremental backups. A full backup makes complete copies of each file. With an incremental backup, only the difference between the current and previous versions of the file are saved. Thus, to retrieve a file from the backup, *Duplicity* must first get the original version of the file, and then apply each change. That approach results in the following issues:

1. The recovery process is slow because the desired file is reconstructed from possibly a large number of change sets, each of which must be downloaded from a remote repository before it can be applied. The change sets are large, so the recovery of even small files can require downloading a large amount of data. It is common that the recovery of a single small file could require many hours.
2. Because the recovery process requires so many steps, it can be fragile. Apparently it keeps all the change sets open during the recovery process, and so the recovery process can fail because the operating system limits how many files you can open at any one time.
3. Generally, when there are problems, you only find them when you try to recover a file. At that point it is too late.
4. Duplicity does not do de-duplication, so if your were to have multiple copies of the same file, or if you moved a file, then you would keep multiple copies of it.

The first two issues can be reduced with frequent full backups, but this greatly increases the space you need to hold your backups.

Borg works in a very different way. When *Borg* encounters a file, it first determines whether it is new or not. The file is determined to be new if the contents of that file do not already exist in the repository, in which case it copies the contents into the repository. Then, either way, it associates a pointer to the file's contents with the filepath. This makes it naturally de-duplicating. When it comes time to recover a file, it simply uses the file path to find the contents. In this way, it only retrieves the data it needs. There is no complicated and fragile process needed to reconstruct the file from a long string of differences.

After living with *Duplicity* for many years, I now find the *Borg* recovery process stunningly fast and extremely reliable. I am completely sold on *Borg* and will never use *Duplicity* again.

CHAPTER

FOUR

TERMINOLOGY

It is helpful to understand two terms that are used used by *Borg* to describe your backups.

repository
: This is the location where all of your files are backed up to. It may be on a local file system or it may be remote, in which case it is accessed using *ssh*.

  A repository consists of a collection of disembodied and deduplicated file contents along with a collection of archives.

archive
: This is a snapshot of the files that existed when a particular backup was run. Basically, it is a collection of file paths along with pointers to the contents of those files.

CHAPTER

FIVE

QUICK TOUR

You must initially describe your repository or repositories to *Emborg*. You do so by adding configuration files to ~/.config/emborg. At least two are required. First is the file that contains settings that are shared between all configurations. This is a Python file located at `~/.config/emborg/settings`. Here is an example:

```
# configurations
configurations = ['root']
default_configurations = 'root'

# things to exclude
exclude_caches = True
exclude_if_present = '.nobackup'
exclude_nodump = True
```

There also must be individual settings files for each backup configuration. They are also a Python files. The above file defines the *root* configuration. The configuration is described in `~/.config/emborg/root`, an example of which is given below. It is designed to back up the whole machine:

```
# destination repository
repository = 'borgbase:backups'
prefix = '{host_name}-{config_name}-'

# directories to back up
src_dirs = ['/']

# directories to exclude
excludes = [
    "/dev",
    "/proc",
    "/run",
    "/sys",
    "/tmp",
    "/var/cache",
    "/var/tmp",
]
```

Since this configuration needs to back up files that may not be accessible by normal users, it should be run by the root user.

Once you have created these files, you can use *Emborg* to perform common tasks that involve your backups.

The first step would be to initialize the remote repository. A repository must be initialized before it can first used. To do so, one uses the *init command*:

```
$ emborg init
```

Once the repository is initialized, it is ready for use. The *create command* creates an archive, meaning that it backs up your current files.

```
$ emborg create
```

Once one or more archives have been created, you can list the available archives using the *list command*.

```
$ emborg list
```

The *manifest or files command* displays all the files in the most recent archive.

```
$ emborg manifest
$ emborg files
```

You can restrict the listing to those files contained in the current working directory using:

```
$ emborg manifest .
```

If you give the name of an archive, it displays all the files in the specified archive.

```
$ emborg manifest -a continuum-2019-04-23T18:35:33
```

Or, you can give a date, in which case the oldest archive created before that date is used.

```
$ emborg manifest -d 2019-04-23
```

You can also specify the date and time relative to the current moment:

```
$ emborg manifest -d 1w
```

The *compare command* allows you to see and manage the differences between your local files and those in an archive. You can compare individual files or entire directories. You can use the date and archive options to select the particular archive to compare against. You can use the interactive version of the command to graphically view changes and merge them back into you local files.

```
$ emborg compare -i doc/thesis
```

The *restore command* restores files or directories in place, meaning it replaces the current version with the one from the archive. You can also use the date and archive options to select the particular archive to draw from.

```
$ cd ~/bin
$ emborg restore accounts.json
```

The *mount command* creates a directory 'BACKUPS' and then mounts an archive or the whole repository on this directory. This allows you to move into the archive or repository, navigating, examining, and retrieving files as if it were a file system. Again, you can use the date and archive options to select the particular archive to mount.

```
$ emborg mount BACKUPS
```

The *umount command* un-mounts the archive or repository after you are done with it.

```
$ emborg umount BACKUPS
```

The *due command* tells you when the last successful backup was performed.

```
$ emborg due
```

The *info command* shows you information about your repository such as where it is located and how large it is.

```
$ emborg info
```

The *help command* shows you information on how to use *Emborg*.

```
$ emborg help
```

There are more commands, but the above are the most commonly used.

CHAPTER

SIX

STATUS

Emborg includes a utility, *emborg_overdue*, that can be run in a cron script on either the client or the server machine that notifies you if your back-ups have not completed successfully in a specified period of time. In addition, *Emborg* can be configured to update monitoring services such as HealthChecks.io with the status of the backups.

CHAPTER

SEVEN

BORG

Borg has more power than what is exposed with *Emborg*. You may use it directly or through the *Emborg* *borg command* when you need that power. More information can be found at Borg.

CHAPTER

EIGHT

PRECAUTIONS

You should assure you have a backup copy of the encryption key and its passphrase in a safe place (run 'borg key export' to extract the encryption keys). This is very important. If the only copy of the encryption credentials are on the disk being backed up and if that disk were to fail you would not be able to access your backups. I recommend the use of SpareKeys as a way of assuring that you always have access to the essential information, such as your *Borg* passphrase and keys, that you would need to get started after a catastrophic loss of your disk.

If you keep the passphrase in an *Emborg* configuration file then you should set the permissions for that file so that it is not readable by others:

```
chmod 600 ~/.config/emborg/*
```

Better is to simply not store the passphrase in *Emborg* configuration files. You can use the *passcommand* setting for this, or you can use Avendesora, which is a flexible password management system. The interface to *Avendesora* is already built in to *Emborg,* but its use is optional (it need not be installed).

It is also best, if it can be arranged, to keep your backups at a remote site so that your backups do not get destroyed in the same disaster, such as a fire or flood, that claims your original files. One option is RSync. Another is BorgBase. I have experience with both, and both seem quite good. One I have not tried is Hetzner.

Finally, it is a good idea to practice a recovery. Pretend that you have lost all your files and then see if you can do a restore from backup. Doing this and working out the kinks before you lose your files can save you if you ever do lose your files.

CHAPTER

NINE

ISSUES

Please ask questions or report problems on GitHub.

CHAPTER

TEN

CONTENTS

10.1 Getting Started

10.1.1 Installing

Many Linux distributions include *Borg* in their package managers. In Fedora it is referred to as *borgbackup*. In this case you would install *borg* by running the following:

```
$ sudo dnf install borgbackup
```

Alternately, you can download a precompiled version from Borg Github Releases, which allows you to install Borg as an unprivileged user. You can do so with following commands (they may need to be adjusted to get the latest version):

```
$ cd ~/bin
$ wget https://github.com/borgbackup/borg/releases/download/1.2.6/borg-linux64
$ wget https://github.com/borgbackup/borg/releases/download/1.2.6/borg-linux64.asc
$ gpg --recv-keys 6D5BEF9ADD2075805747B70F9F88FB52FAF7B393
$ gpg --verify borg-linux64.asc
$ rm borg-linux64.asc
$ chmod 755 borg-linux64
```

Finally, you can install it using pip:

```
$ pip install --user borgbackup
```

Download and install *Emborg* as follows (requires Python3.6 or better):

```
$ pip install --user emborg
```

Or, if you want the development version, use:

```
$ git clone https://github.com/KenKundert/emborg.git
$ pip install --user ./emborg
```

You may also need to install and configure either a notification daemon or a mail daemon. This allows errors to be reported when you are not running *Emborg* in a terminal. More information can be found by reading about the *notifier* and *notify* *Emborg* settings.

10.1.2 Configuring Emborg to Backup A Home Directory

The basic idea behind *Emborg* is that you place all information relevant to your backups in two configuration files, which allows you to use *Emborg* to perform tasks without re-specifying that information. Emborg allows you to have any number of setups, which you might want if you wanted to backup to multiple repositories for redundancy or if you want to use different rules for different sets of files. Regardless, you use a separate configuration for each set up, plus there is a common configuration file shared by all setups. You are free to place most settings in either file, which ever is most convenient. All the configuration files are placed in ~/.config/emborg. If you run *Emborg* without creating your configuration files, *Emborg* will create some starter files for you. A configuration is specified using Python, thus the content of these files is formatted as Python code and is read by a Python interpreter.

As a demonstration on how to configure *Emborg*, imagine wanting to back up your home directory in two ways. First, you want to backup the files to an off-site server. Here the expectation is that you would backup once a day on average and you would do so interactively so that you can choose an appropriate time. Second, you have some free space on your machine that you would like to dedicate to recent snapshots of your files. The idea is that you find that you occasionally overwrite or delete files that you just spent time creating, and you want to run local backups every 10-15 minutes so that you can easily recover these files. To accomplish these two things, you need three configuration files.

Shared Settings

The first file is the shared configuration file:

```
configurations = 'backups snapshots'
default_configuration = 'backups'
```

This is basically the minimum you can give. Your two configurations are listed in *configurations*. It could be a list of strings, but you can also give a single string, in which case the string is split on white space. Then you specify your default configuration. In this example *backups* is to be run interactively and *snapshots* is to be run on a schedule by *cron*, so the default is set to *backups* to make it easier to run interactively.

Configuration for a Remote Repository: *backups*

The second file is the configuration file for *backups*:

```
repository = 'backups:archives'
prefix = '{host_name}-'
encryption = 'keyfile'
passphrase = 'crone excess mandate bedpost'

src_dirs = '~'
excludes = '''
    ~/.cache
    **/*~
    **/.git
    **/__pycache__
    **/.*.swp
'''
exclude_if_present = '.nobackup'

check_after_create = 'latest'
prune_after_create = True
compact_after_delete = True
keep_daily = 7
```

(continues on next page)

(continued from previous page)

```
keep_weekly = 4
keep_monthly = 12
keep_yearly = 2
```

This configuration assumes that you have a *backups* entry in your SSH config file that contains the appropriate user name, host name, port number, and such for the server that contains your remote repository. It also assumes that you have shared an SSH key with this server so you do not need to specify a password each time you back up, and that that key is pre-loaded into your SSH agent. The repository is actually in the *archives* directory on that server, and each back-up archive will be prefixed with your local host name, allowing you to share this repository with other machines.

You specify what to backup using *src_dirs* and what not to backup using *excludes*. Nominally both *src_dirs* and *excludes* take lists of strings, but you can also specify them using a single string, in which case the strings are broken into individual lines, any blank lines or lines that begin with # are ignored, and then the white space is removed from the front and back of each line.

This configuration file ends with settings that tell *Emborg* to run *check* and *prune* operations after creating a backup, and it gives the desired prune schedule.

This is just an example, and a rather minimal one at that. You should not use it without understanding each of the settings. The *encryption* setting is a particularly important one for you to understand and set properly. More comprehensive information about configuring *Emborg* can be found in the section on *Configuring*.

With this configuration, you can now initialize your repository and use it to perform backups. If the repository does not yet exist, initialize it using:

```
$ emborg init
```

Then perform a back up using:

```
$ emborg create
```

or simply:

```
$ emborg
```

This works because *create* is the default action and *backups* is the default configuration.

Then, you can convince yourself it is working as expected by moving a directory out of the way and using *Emborg* to restore it:

```
$ mv bin bin-saved
$ emborg restore bin
```

Configuration for a Local Repository: *snapshots*

The third file is the configuration file for *snapshots*:

```
repository = '/mnt/snapshots/{user_name}'
prefix = '{config_name}-'
encryption = 'none'

src_dirs = '~'
excludes = '''
    ~/.cache
```

(continues on next page)

(continued from previous page)

```
    **/*~
    **/.git
    **/__pycache__
    **/.*.swp
'''
prune_after_create = True
compact_after_delete = True
keep_within = '1d'
```

In this case the repository is on the local machine and it is not encrypted. It again backs up your home directory, but for this configuration the archives are only kept for a day.

The repository must be initialized before it can be used:

```
$ emborg -c snapshots init
```

Here the desired configuration was specified because it is not the default. Now, a *cron* entry can be created using `crontab -e` that creates a snapshot every 10 minutes:

```
*/10 * * * *  emborg --config snapshots --mute create
```

Once it has run, you can pull a file from the latest snapshot using:

```
$ emborg -c snapshots restore passwords.gpg
```

Overdue Backups

Emborg allows you to easily determine when your files were last backed up using:

```
$ emborg due
```

However, you must remember to run this command. *Emborg* also provides *emborg-overdue* to provide automated reminders. You configure *emborg-overdue* using a configuration file: ~/.config/emborg/overdue.conf. For example:

```
default_maintainer = 'me@mydomain.com'
dumper = 'me@mydomain.com'
default_max_age = 36 # hours
root = '~/.local/share/emborg'
repositories = [
    dict(host='laptop (snapshots)', path='snapshots.lastbackup', max_age=0.2),
    dict(host='laptop (backups)', path='backups.lastbackup'),
]
```

Then you would configure *cron* to run *emborg-overdue* using something like:

```
00 * * * * ~/.local/bin/emborg-overdue --quiet --mail
```

This runs *emborg-overdue* every hour on the hour, and it reports any delinquent backups by sending mail to the appropriate maintainer (the message is sent from the *dumper*). You can specify any number of repositories to check, and for each repository you can specify *host* (a descriptive name), *path* (the path to the repository from the *root* directory, a *max_age* in hours, and a *maintainer*. You can also specify defaults for the *maintainer* and *max_age*. When run, it checks the age of each repository and sends email to the appropriate maintainer if it exceeds the maximum allowed age.

In this example the actual repository is not checked directly, rather the *lastbackup* file is checked. This is a file that is updated by *Emborg* after every back up. This file is found in the *Emborg* output directory. Every time *Emborg* runs it creates a log file that can also be found in this directory. That logfile can be viewed directly, or you can view it using the *log* command:

```
$ emborg log
```

10.1.3 Configuring Emborg to Backup an Entire Machine

The primary difference between this example and the previous is that *Emborg* needs to be configured and run by *root*. This allows all the files on the machine to be backed up regardless of who owns them. Other than being root, the mechanics are very much the same.

To start, run *emborg* as root to create the initial configuration files:

```
# emborg
```

This creates the /root/.config/emborg directory in the root account and populates it with three files: *settings*, *root*, *home*. You can delete *home* and remove the reference to it in *settings*, leaving only:

```
configurations = 'root'
default_configuration = 'root'
```

This assumes that most of the settings will be placed in *root*:

```
repository = 'backups:backups/{host_name}'
prefix = '{config_name}-'
passphrase = 'western teaser landfall spearhead'
encryption = 'repokey'

src_dirs = '/'
excludes = '''
    /dev
    /home/*/.cache
    /proc
    /root/.cache
    /run
    /sys
    /tmp
    /var
'''

check_after_create = 'latest'
compact_after_delete = True
prune_after_create = True
keep_daily = 7
keep_weekly = 4
keep_monthly = 12
```

Again, this is a rather minimal example. In this case, *repokey* is used as the encryption method, which is only suitable if the repository is on a server you control.

When backing up the root file system it is important to exclude directories that cannot or should not be backed up. Those include: /dev, /proc, /run, /sys, and /tmp.

As before you need to initialize the repository before it can be used:

```
# emborg init
```

To assure that the backups are run daily, the following is added to /etc/cron.daily/emborg:

```
#/bin/sh
# Run root backups

emborg --mute --config root create
```

This is preferred for laptops because cron.daily is guaranteed to run each day as long as machine is turned on for any reasonable length of time.

10.2 Commands

You invoke *Emborg* from your shell by entering a line of the form:

```
$ emborg [global-options] <command> [command-options]
```

Details about the options and commands can be accessed with:

```
$ emborg help
```

or:

```
$ emborg help <command>
```

The available commands are:

borg
run a raw borg command

breaklock
breaks the repository and cache locks

check
checks the repository and its archives

compact
compact segment files in the repository

compare
compare local files with those in an archive

configs
list available backup configurations

create
create an archive of the current files

delete
delete an archive currently contained in the repository

diff
show the differences between two archives

due
days since last backup

extract
recover file or files from archive

help
give information about commands or other topics

info
print information about a backup

init
initialize the repository

list
list the archives currently contained in the repository

log
print logfile for the last emborg run

manifest
list the files contained in an archive

mount
mount a repository or archive

prune
prune the repository of excess archives

restore
recover file or files from archive in place

settings
list settings of chosen configuration

umount
un-mount a previously mounted repository or archive

version
display emborg version

These commands are described in more detail below. Not everything is described here. Run `emborg help <cmd>` for the details.

10.2.1 Exit Status

Emborg returns with an exit status of 0 if it completes without issue. It returns with an exit status of 1 if was able to terminate normally but some exceptional condition was encountered along the way. For example, if the *compare* or *diff* detects a difference or if *due* command detects the backups are overdue, a 1 is returned. In addition, 1 is returned if *Borg* detects an error but is able to complete anyway. However, if *Emborg* or *Borg* suffers errors and cannot complete, 2 is returned.

10.2.2 Borg

Runs raw *Borg* commands. Before running the passphrase or passcommand is set. Also, if `@repo` is found on the command line, it is replaced by the path to the repository.

```
$ emborg borg key export @repo key.borg
$ emborg borg list @repo::root-2020-04-11T23:38:37
```

Emborg runs the *Borg* command from *working_dir* if it is specified and `/` if not.

10.2.3 BreakLock

This command breaks the repository and cache locks. Use carefully and only if no *Borg* process (on any machine) is trying to access the Cache or the Repository.

```
$ emborg break-lock
$ emborg breaklock
```

10.2.4 Check

Check the integrity of the repository and its archives. The most recently created archive is checked if one is not specified unless `--all` is given, in which case all archives are checked.

The `--repair` option attempts to repair any damage found. Be aware that the –repair option is considered a dangerous operation that might result in the complete loss of corrupt archives. It is recommended that you create a backup copy of your repository and check your hardware for the source of the corruption before using this option.

10.2.5 Compact

This command frees repository space by compacting segments.

Use this regularly to avoid running out of space, however you do not need to it after each *Borg* command. It is especially useful after deleting archives, because only compaction really frees repository space.

Requires Borg version 1.2 or newer. Prior to version 1.2 the compact functionality was part of the *Borg prune* command. As of version 1.2 this functionality was split into its own command.

If you set *compact_after_delete* *Emborg* automatically runs this command after every use of the *delete* and *prune* commands.

10.2.6 Compare

Reports and allows you to manage the differences between your local files and those in an archive. The base command simply reports the differences:

```
$ emborg compare
```

The `--interactive` option allows you to manage those differences. Specifically, it will open an interactive file comparison tool that allows you to compare the contents of your files and copy differences from the files in the archive to your local files:

```
$ emborg compare -i
```

You can specify the archive by name or by date or age. If you do not you will use the most recent archive:

```
$ emborg compare -a continuum-2020-12-04T17:41:28
$ emborg compare -d 2020-12-04
$ emborg compare -d 1w
```

You can specify a path to a file or directory to compare, if you do not you will compare the files and directories of the current working directory.

```
$ emborg compare tests
$ emborg compare ~/bin
```

This command uses external tools to view and manage the differences. Before it can be used it must be configured to use these tools, which is done with the *manage_diffs_cmd* and *report_diffs_cmd* settings. In addition, the *default_mount_point* must be configured. The *manage_diffs_cmd* is used if the `--interactive` (or `-i`) option is given, and *report_diffs_cmd* otherwise. However, if only one is given it is used in both cases. So, if you find that you only want to use the interactive tool to view and manage your differences, you can simply not specify *report_diffs_cmd*, which would eliminate the need to specify the `-i` option.

The command operates by mounting the desired archive, performing the comparison, and then unmounting the directory. Problems sometimes occur that can result in the archive remaining mounted. In this case you will need to resolve any issues that are preventing the unmounting, and then explicitly run the *umount command* before you can use this *Borg* repository again.

This command differs from the *diff command* in that it compares local files to those in an archive where as *diff* compares the files contained in two archives.

10.2.7 Configs

List the available backup configurations. Each configuration corresponds to a settings file in your configuration directory (~/.config/emborg). Settings common to all your configurations should be placed in ~/.config/emborg/settings. You can see available configurations using:

```
$ emborg configs
```

To run a command on a specific configuration, add –config=<cfg> or -c cfg before the command. For example:

```
$ emborg -c home create
```

10.2.8 Create

This creates an archive in an existing repository. An archive is a snapshot of your files as they currently exist. Borg is a de-duplicating backup program, so only the changes from the already existing archives are saved.

```
$ emborg create
```

Before creating your first archive, you must use the *init* command to initialize your repository.

This is the default command, so you can create an archive with simply:

```
$ emborg
```

If the backup seems to be taking a long time for no obvious reason, run the backup in verbose mode:

```
$ emborg -v create
```

This can help you understand what is happening.

Emborg runs the *create* command from *working_dir* if it is specified and current directory if not.

10.2.9 Delete

Delete one or more archives currently contained in the repository:

```
$ emborg delete continuum-2020-12-05T19:23:09
```

If no archive is specified, the latest is deleted.

The disk space associated with deleted archives is not reclaimed until the *compact* command is run. You can specify that a compaction is performed as part of the deletion by setting *compact_after_delete*. If set, the `--fast` flag causes the compaction to be skipped. If not set, the `--fast` flag has no effect.

Specifying `--repo` results in the entire repository being deleted. Unlike with *borg* itself, no warning is issued and no additional conformation is required.

10.2.10 Diff

Shows the differences between two archives:

```
$ emborg diff continuum-2020-12-05T19:23:09 continuum-2020-12-04T17:41:28
```

You can constrain the output listing to only those files in a particular directory by adding that path to the end of the command:

```
$ emborg diff continuum-2020-12-05T19:23:09 continuum-2020-12-04T17:41:28 .
```

This command differs from the *compare command* in that it only reports a list of files that differ between two archives, whereas *compare* shows how local files differ from those in an archive and can show you the contents of those files and allow you interactively copy changes from the archive to your local files.

10.2.11 Due

When run with no options it indicates when the last backup and squeeze operations were performed. A backup operation is the running of the *create* command. A squeeze operation is the running of both the *prune* and *compact* commands. The time to the latest squeeze operation is the time to the older of the most recent *prune* or *compact* commands. For example:

```
$ emborg due
home: 11 hours since last backup.  2 weeks since last squeeze.
```

Adding the –backup-days option or –squeeze-days results in the message only being printed if the backup or squeeze has not been performed within the specified number of days. If both are specified and both limits are violated, only the backup violation is reported as it is considered the most urgent.

Adding the –email option results in the message being sent to the specified address rather than printed. This allows you to run the *due* command from a cron script in order to send your self reminders to do a backup if one has not occurred for a while. It is often run with the –no-log option to avoid replacing the log file with one that is inherently uninteresting:

```
$ emborg --no-log due --backup-days 1 --backup-days 7 --email me@mydomain.com
```

You can specify a specific message to be printed with –message. In this case, the following replacements are available:

{action}:
Replaced with the type of operation reported on. It is either *backup* or *squeeze*.

{config}:
Replaced with the name of the configuration being reported on.

{cmd}:
Replaced with the name of the command being reported on. It can be *create*, *prune* or *compact*. It will be *create* if reporting on a backup operation, and either *prune* or *compact* if reporting on a squeeze operation, depending on which is older.

{days}:
Replaced by the number of days since the last backup or squeeze. You can add floating-point format codes to specify the resolution used. For example: {days:.1f}.

{elapsed}:
Replaced with a humanized description of how long it has been since the last backup.

So `--message '{elapsed} since last {action} of {config}.'` might produce something like this:

```
12 hours since last backup of home.
```

With composite configurations the message is printed for each component config unless –oldest is specified, in which case only the oldest is displayed.

10.2.12 Extract

You extract a file or directory from an archive using:

```
$ emborg extract home/shaunte/bin
```

Use manifest to determine what path you should specify to identify the desired file or directory. You can specify more than one path. Usually, they will be paths that are relative to /, thus the paths should look like absolute paths with the leading slash removed. The paths may point to directories, in which case the entire directory is extracted. It may also be a glob pattern.

By default, the most recent archive is used, however, if desired you can explicitly specify a particular archive. For example:

```
$ emborg extract --archive continuum-2020-12-05T12:54:26 home/shaunte/bin
```

Alternatively you can specify a date or date and time. If only the date is given the time is taken to be midnight. The oldest archive that is younger than specified date and time is used. For example:

```
$ emborg extract --date 2021-04-01 home/shaunte/bin
$ emborg extract --date 2021-04-01T15:30 home/shaunte/bin
```

Alternatively, you can specify the date in relative terms:

```
$ emborg extract --date 3d  home/shaunte/bin
```

In this case 3d means 3 days. You can use s, m, h, d, w, M, and y to represent seconds, minutes, hours, days, weeks, months, and years.

Finally, if you specify a simple number, it is taken to be the index of the desired archive, where 0 represents the most recent, 1 the next most recent, etc.

```
$ emborg extract --date 3  home/shaunte/bin
```

The extracted files are placed in the current working directory with the original hierarchy. Thus, the above commands create the directory:

```
./home/shaunte/bin
```

See the *restore* command as an alternative to *extract* that replaces the existing files rather than simply copying them into the current directory.

10.2.13 Help

Show information about Emborg:

```
$ emborg help
```

You can ask for help on a specific command or topic with:

```
$ emborg help <topic>
```

For example:

```
$ emborg help extract
```

10.2.14 Info

This command prints out the locations of important files and directories.

```
$ emborg info
```

You can also get information about a particular archive.

```
$ emborg info home-2022-11-03T23:07:25
```

10.2.15 Init

Initializes a Borg repository. This must be done before you create your first archive.

```
$ emborg init
```

10.2.16 List

List available archives.

```
$ emborg list
```

10.2.17 Log

Show the log from the previous run.

```
$ emborg log
```

Most commands save a log file, but some do not. Specifically, *configs*, *due*, *help*, *log*, *settings* and *version* do not. Additionally, no command will save a log file if the `--no-log` command line option is specified. If you need to debug a command that does not normally generate a log file and would like the extra detail that is normally included in the log, specify the `--narrate` command line option.

If you wish to access the log files directly, they reside in `~/.local/share/emborg`.

10.2.18 Manifest

Once a backup has been performed, you can list the files available in your archive using:

```
$ emborg manifest
```

You specify a path. If so, the files listed are those contained within that path. For example:

```
$ emborg manifest .
$ emborg manifest -R .
```

The first command lists the files in the archive that were originally contained in the current working directory. The second lists the files that were in specified directory and any sub directories.

If you do not specify an archive, as above, the latest archive is used.

You can explicitly specify an archive:

```
$ emborg manifest --archive continuum-2021-04-01T12:19:58
```

Or you choose an archive based on a date and time. The oldest archive that is younger than specified date and time is used.

```
$ emborg manifest --date 2021-04-01
$ emborg manifest --date 2021-04-01T12:45
```

You can also specify the date in relative terms:

```
$ emborg manifest --date 1w
```

where s, m, h, d, w, M, and y represents seconds, minutes, hours, days, weeks, months, and years.

Finally, if you specify a simple number, it is taken to be the index of the desired archive, where 0 represents the most recent, 1 the next most recent, etc.

```
$ emborg manifest --date 7
```

The *manifest* command provides a variety of sorting and formatting options. The formatting options are under the control of the *manifest_formats* setting. For example:

```
$ emborg manifest
```

This outputs the files in the order and with the format produced by Borg. If a line is green if the corresponding file is healthy, and if red it is broken (see Borg list command for more information on broken files).

```
$ emborg manifest -l
$ emborg manifest -n
```

These use the Borg order but change the amount of information shown. With `-l` the *long* format is used, which by default contains the size, the date, and the path. With `-n` the *name* is used, which by default contains only the path.

Finally:

```
$ emborg manifest -S
$ emborg manifest -D
```

The first sorts the files by size. It uses the *size* format, which by default contains only the size and the path. The second sorts the files by modification date. It uses the *date* format, which by default contains the day, date, time and the path. More choices are available; run `emborg help manifest` for the details.

You can use `files` as an alias for `manifest`:

```
$ emborg files
```

10.2.19 Mount

Once a backup has been performed, you can mount it and then look around as you would a normal read-only filesystem.

```
$ emborg mount backups
```

In this example, *backups* acts as a mount point. If it exists, it must be a directory. If it does not exist, it is created.

If you do not specify a mount point, the value of *default_mount_point* setting is used if set.

If you do not specify an archive, as above, the most recently created archive is mounted.

You can explicitly specify an archive:

```
$ emborg mount --archive continuum-2015-04-01T12:19:58 backups
```

You can mount the files that existed on a particular date using:

```
$ emborg mount --date 2021-04-01 backups
$ emborg mount --date 2021-04-01T18:30 backups
```

If the time is not given, it is taken to be midnight.

You can also specify the date in relative terms:

```
$ emborg mount --date 1w backups
```

where s, m, h, d, w, M, and y represents seconds, minutes, hours, days, weeks, months, and years.

Finally, if you specify a simple number, it is taken to be the index of the desired archive, where 0 represents the most recent, 1 the next most recent, etc.

```
$ emborg mount --date 7 backups
```

When a date is given, the oldest archive that is younger than the specified date or time is used.

Finally, you can mount all the available archives:

```
$ emborg mount --all backups
```

You will need to un-mount the repository or archive when you are done with it. To do so, use the *umount* command.

10.2.20 Prune

Prune the repository of excess archives. You can use the *keep_within*, *keep_last*, *keep_minutely*, *keep_hourly*, *keep_daily*, *keep_weekly*, *keep_monthly*, and *keep_yearly* settings to control which archives should be kept. At least one of these settings must be specified to use *prune*:

```
$ emborg prune
```

The *prune* command deletes archives that are no longer needed as determined by the prune rules. However, the disk space is not reclaimed until the *compact* command is run. You can specify that a compaction is performed as part of the prune by setting *compact_after_delete*. If set, the `--fast` flag causes the compaction to be skipped. If not set, the `--fast` flag has no effect.

10.2.21 Restore

This command is very similar to the *extract* command except that it is meant to be run in place. Thus, the paths given are converted to absolute paths and then the borg *extract* command is run from the root directory (/) so that the existing files are replaced by the extracted files.

For example, the following commands restore your .bashrc file:

```
$ cd ~
$ emborg restore .bashrc
```

Emborg runs the *restore* command from *working_dir* if it is specified and the current directory if not.

By default, the most recent archive is used, however, if desired you can explicitly specify a particular archive. For example:

$ emborg restore –archive continuum-2020-12-05T12:54:26 resume.doc

Or you choose an archive based on a date and time. The oldest archive that is younger than specified date and time is used.

$ emborg restore –date 2021-04-01 resume.doc $ emborg restore –date 2021-04-01T18:30 resume.doc

Or you can specify the date in relative terms:

$ emborg restore –date 3d resume.doc

In this case 3d means 3 days. You can use s, m, h, d, w, M, and y to represent seconds, minutes, hours, days, weeks, months, and years.

Finally, if you specify a simple number, it is taken to be the index of the desired archive, where 0 represents the most recent, 1 the next most recent, etc.

```
$ emborg restore --date 7 resume.doc
```

This command is very similar to the *extract* command except that it is meant to replace files in place. It also takes similar options.

10.2.22 Settings

This command displays all the settings that affect a backup configuration.

```
$ emborg settings
```

Add `--all` option to list out all available settings and their descriptions rather than the settings actually specified and their values.

10.2.23 Umount

Un-mount a previously mounted repository or archive:

```
$ emborg umount backups
$ rmdir backups
```

where *backups* is the existing mount point.

If you do not specify a mount point, the value of *default_mount_point* setting is used if set.

10.2.24 Version

Prints the *Emborg* version.

```
$ emborg version
```

10.3 Configuring

Typically the settings files go in the default location for configuration files on your system. On Linux systems, that location is ~/.config/emborg. Other systems use more awkward locations, so while *Emborg* creates initial versions in the default location, you are free to move them to ~/.config/emborg if you prefer. *Emborg* always checks for the files in ~/.config/emborg if it exists before looking in the default location for your system.

You need a shared settings file and then one file for each backup configuration you need. Except for *configurations* and *default_configuration* any setting may be placed in either the shared file or the configuration specific file. If a setting is found in both files, the version in the configuration specific file dominates.

You can get a complete list of available configuration settings by running:

```
$ emborg settings --available
```

10.3.1 Shared Settings

Shared settings go in ~/.config/emborg/settings. This is a Python file that contains values shared by all of your configurations. It might look like the following:

```
default_configuration = 'home'          # default backup configuration
configurations = 'home websites'        # available backup configurations
avendesora_account = 'borg-backup'      # Avendesora account name (holds passphrase for␣
↪encryption key)
passphrase = None                       # passphrase to use (if specified, Avendesora is␣
↪not used)
encryption = 'keyfile'                  # encryption method
prune_after_create = True               # run prune as the last step of an archive creation
check_after_create = 'latest'           # run check as the last step of an archive creation
#notify = "me@mydomain.com"              # email address to notify when things go wrong
notifier = 'notify-send -u normal {prog_name} "{msg}"'
                                        # program used to send realtime notifications
                                        # generally you use notify or notifier, but not␣
↪both
                                        # use notifier for interactive backups
                                        # and notify for scheduled backups
                                        # notification program
upload_ratelimit = 2000                 # bandwidth limit in kbps
umask = '077'                           # umask to use when creating the archives
repository = 'archives:/mnt/backups/{host_name}/{user_name}/{config_name}'
                                        # remote directory for repository
archive = '{host_name}-{{now}}'         # naming pattern used for the archives
    # May contain {<name>} where <name> may be any of host_name, user_name,
    # prog_name config_name, or any of the user specified settings.
    # Double up the braces to specify parameters that should be interpreted
    # by borg rather than by emborg.
exclude_caches = True                   # do not backup directories that contain CACHEDIR.
↪TAG
exclude_if_present = '.nobackup'        # do not backup directories containing this file
keep_within = '1d'                      # keep all archives within this time interval
keep_hourly = '48'                      # number of hourly archives to keep
keep_daily = '7'                        # number of daily archives to keep
keep_weekly = '4'                       # number of weekly archives to keep
keep_monthly = '12'                     # number of weekly archives to keep
keep_yearly = '2'                       # number of weekly archives to keep
```

If you encrypt your backups, you can specify the encryption key in this file as *passphrase*. In this case, you should be careful to assure the file is not readable by others (chmod 600 settings). Alternatively, you can use *passcommand*, which runs a command that returns your pass phrase. Finally, you can use Avendesora to securely hold your key by specifying the Avendesora account name of the key to *avendesora_account*.

This example assumes that there is one backup configuration per repository. You can instead have more than one configurations share a single repository by adjusting *repository* and adding *glob_archives* like so:

```
repository = 'archives:/mnt/backups/{host_name}/{user_name}'
glob_archives = '{config_name}-*'
```

In this case several backup configurations would deposit archives into a single directory, allowing them to reduce the total space required to hold the archives if there are shared files between the configurations. The *glob_archives* setting is required to allow each backup configuration to recognize its own archives. All archive names that match the glob

string associate with this configuration.

10.3.2 Configurations

Each backup configuration must have a settings file in ~/.config/emborg. The name of the file is the name of the backup configuration. It might look like the following:

```
src_dirs = '~'           # absolute paths to directories to be backed up
excludes = """
    ~/tmp
    ~/**/.hg
    ~/**/.git
    ~/**/*.pyc
    ~/**/.*.swp
    ~/**/.*.swo
"""                      # list of glob strings of files or directories to skip
one_file_system = False  # okay to traverse filesystems

# commands to be run before and after backups (run from working directory)
run_before_first_backup = """
    # remove the detritus before backing up
    ~/bin/clean-home >& {log_dir}/clean-home.log
"""
run_after_last_backup = """
    # rebuild my documentation, it was deleted by clean-home
    ~/bin/rebuild-documentation > /dev/null
"""

# if set, this file or these files must exist or backups will quit with an error
must_exist = '~/doc/thesis'
```

String values may incorporate other string valued settings. Use braces to interpolate another setting. In addition, you may interpolate the configuration name ('config_name'), the host name ('host_name'), the user name ('user_name'), Emborg's program name ('prog_name'), your home directory ('home_dir'), the configuration directory ('config_dir') or the output directory ('log_dir'). An example of this is shown in both *repository* and *archive* above. Doubling up the braces acts to escape them. In this way you gain access to *Borg* placeholders. *archive* shows an example of that. Interpolation is not performed on any setting whose name is given in *do_not_expand*.

Settings that take lists of strings can be specified as a single multi-line string where one item is given per line. Lines that begin with # are ignored, as are empty lines. For example:

```
excludes = """
    # these directories would be problematic if backed up
    /dev
    /proc

    # these directories contain largely derived files which can be recreated
    /run
    /sys
    /tmp
    /var
"""
```

10.3.3 Paths

When *Borg* places files into a repository, it always uses relative paths. However, you may specify them either using relative paths or absolute paths. *Borg* starts backing up from the recursion roots. These are directories that you specify to *src_dirs* or using the `R` key in *patterns* or *patterns_from*. Within a recursion root you can specify particular paths to exclude and within those you can specify particular files to include. This is done using *excludes* and *exclude_from* and using the path keys (`+`, `-`, `!`) in *patterns* and *patterns_from*. When you use a relative path to specify a recursion root then you should also use relative paths for its include and exclude paths. Similarly, if you use an absolute path for the recursion root then you should also use absolute paths for its include and exclude paths. *Borg* is okay with you having some recursion roots specified with relative paths and some with absolute paths, but this confuses *Emborg* when it comes time to extract or restore files from your repository. With *Emborg*, all of your recursive roots must either be specified using relative paths or they must all be specified with absolute paths.

If you specify absolute paths, *Borg* converts them to relative paths as it inserts them into the repository by stripping off the leading `/` from the path. If you specify relative paths, it inserts them as is. When using *Borg* directly, the relative paths would be relative to the directory where *borg create* is invoked. For this reason, *borg create* must always be invoked from the same directory when using relative paths. To make this work, *Emborg* internally changes to *working_dir* before running *borg create*. Thus, if you choose to use relative paths, you should also specify *working_dir*, which should be specified with an absolute path. For example:

```
working_dir = '~'
src_dirs = '.'
excludes = """
    .cache
    *~
"""
```

If you do not specify *working_dir*, it defaults to `/`.

Other than paths to include files, all relative paths specified in your configuration are relative to *working_dir*. This can be confusing, so it is recommended that all paths in your configuration, other than those being passed directly to *Borg* should be given using absolute paths. This includes settings such as *default_mount_point*, *must_exist*, *patterns_from*, and *exclude_from*.

Paths specified directly to *Emborg* are processed and any leading tildes (~) are expanded to the appropriate user's home directory. However, paths specified in *exclude_from* and *patterns_from* files are processed directly by *Borg*, which does not expand tildes to a user's home directory.

10.3.4 Includes

Any settings file may include the contents of another file by using *include*. You may either specify a single include file as a string or a collection as a list of strings or a multi-line string. For example:

```
include = 'file-to-include'
```

or:

```
include = """
    first-file-to-include
    second-file-to-include
"""
```

If you specify a relative path for an include file, it it relative to the file that includes it.

10.3.5 Composite Configurations

It is possible to define composite configurations that allow you to run several configurations at once. This might be useful if you want to backup to more than one repository for redundancy. Or perhaps you have files that benefit from different prune schedules.

As an example, consider having three configurations that you would like to run all at once. You can specify these configurations as follows:

```
configurations = 'home lamp data all=home,lamp,data'
```

In this case *home*, *lamp*, and *data* are simple configurations and *all* is a composite configuration. *home*, *lamp*, and *data* would have configuration files whereas *all* would not. The composite configuration should be specified without spaces.

You can run a specific configuration with:

```
$ emborg -c home extract ~/bin
```

You can run all three configurations with:

```
$ emborg -c all create
```

Only certain commands support composite configurations, and if a command does support composite configurations it may either apply each subconfig in sequence, or only the first subconfig.

Command	Response to Composite Config
borg	error
breaklock	error
check	run on each subconfig
configs	does not use any configurations
create	run on each subconfig
delete	error
diff	error
due	run on each subconfig
extract	run only on first subconfig
help	does not use any configurations
info	run on each subconfig
initialize	run on each subconfig
list	run only on first subconfig
log	run on each subconfig
manifest	run only on first subconfig
mount	run only on first subconfig
prune	run on each subconfig
restore	run only on first subconfig
settings	error
umount	run only on first subconfig
version	does not use any configurations

10.3.6 Patterns

Patterns are a relatively new feature of *Borg*. They are an alternate way of specifying which files are backed up, and which are not. Patterns can be specified in conjunction with, or instead of, *src_dirs* and *excludes*. One powerful feature of patterns is that they allow you to specify that a directory or file should be backed up even if it is contained within a directory that is being excluded.

An example that uses *patterns* in lieu of *src_dirs* and *excludes* is:

```
patterns = """
    R /
    + /home/susan
    - /home
    - /dev
    - /opt
    - /proc
    - /run
    - /sys
    - /tmp
    - /var
"""
```

In this example, `R` specifies a root, which would otherwise be specified to *src_dirs*. `+` specifies path that should be included in the backups and `-` specifies a path that should be excluded. With this example, Susan's home directory is included while all other home directories are not. In cases such as this, the subdirectory to include must be specified before the directory that contains it is excluded. This is a relatively simple example, additional features are described in the Borg patterns documentation.

10.3.7 Archive Retention

You use the retention limits (the `keep_X` settings) to specify how long to keep archives after they have been created. A good description of the use of these settings can be found on the Borg Prune Command page.

Generally you want to thin the archives out more and more as they age. When choosing your retention limits you need to consider the nature of the files you are archiving. Specifically you need to consider how often the files change, whether you would want to recover prior versions of the files you keep and if so how many prior versions are of interest, and how long precious files may be missing or damaged before you notice that they need to be restored.

If files are changing all the time, long high retention limits result in high storage requirements. If you want to make sure you retain the latest version of a file but you do not need prior versions, then you can reduce your retention limits to reduce your storage requirements. For example, consider a directory of log files. Log files generally change all the time, but they also tend to be cumulative, meaning that the latest file contains the information contained in prior versions of the same file, so keeping those prior versions is of low value. In this situation using "*keep_last N*" where *N* is small is a good approach.

Now consider a directory of files that should be kept forever, such as family photos or legal documents. The loss of these files due to disk corruption or accidental deletion might not be noticed for years. In this case you would want to specify "*keep_yearly N*" where *N* is large. These files never change, so the de-duplication feature of *Borg* avoids growth in storage requirements despite high retention limits.

You cannot specify retention limits on a per file or per directory basis within a single configuration. Instead, if you feel it is necessary, you would create individual configurations for files with different retention needs. For example, as a system administrator you might want to create separate configurations for operating system files, which tend to need low retention limits, and users home directories, which benefit from longer retention limits.

Remember that your retention limits are not enforced until you run the *prune command*. Furthermore, with *Borg 1.2* and later, after running the *prune command*, the disk space is not reclaimed until you run the *compact command*. You can automate pruning and compaction using the *prune_after_create* and *compact_after_delete* settings.

10.3.8 Confirming Your Configuration

Once you have specified your configuration you should carefully check it to make sure you are backing up the files you need and not backing up the files you don't need. It is important to do this in the beginning, otherwise you might find your self with a bloated repository that does not contain the files you require.

There are a number of ways that *Emborg* can help you check your work.

1. You can run `emborg settings` to see the values used by *Emborg* for all settings.
2. You can use *Borg*'s `--dry-run` option to perform a practice run and see what will happen. For example:

   ```
   $ emborg --dry-run create --list
   ```

   will show you all of the files that are to be backed up and which of those files have changed since the last time you created an archive.
3. After running *Emborg* you can run `emborg log` to see what *Emborg* did in detail and what it asked *Borg* to do. The log contains the full *Borg* command invocation and *Borg*'s response.
4. Once you have created your repository and created your first archive, you can use the `--sort-by-size` option of the *manifest command* to find the largest files that were copied into the repository. If they are not needed, you can add them to your exclude list, delete the archive, and then recreate the archive, this time without the large unnecessary files.

10.3.9 Emborg Settings

These settings control the behavior of *Emborg*.

archive

archive is a template that specifies the name of each archive. A typical value might be:

```
archive = '{config_name}-{{now}}'
```

Emborg examines the string for names within a single brace-pair and replaces them with the value specified by the name. Names within double-brace pairs are interpreted by *Borg*.

More than one backup configuration can share the same repository. This allows *Borg*'s de-duplication feature to work across all configurations, resulting in less total space needed for the combined set of all your archives. In this case you must also set the *glob_archives* setting so that each backup configuration can recognize its own archives. It is used by the *Check*, *Delete*, *Info*, *List*, *Mount*, and *Prune* commands to filter out archives not associated with the desired backup configuration.

The *archive* setting should include *{{now}}* so each archive has a unique name, however you can customize how *now* is expanded. For example, you can reduce the length of the timestamp using:

```
archive = '{host_name}-{{now:%Y%m%d}}'
```

However, you should be aware that by including only the date in the archive name rather than the full timestamp, you are limiting yourself to creating one archive per day. A second archive created on the same day simply writes over the previous archive.

avendesora_account

An alternative to *passphrase*. The name of the Avendesora account used to hold the passphrase for the encryption key. Using *Avendesora* keeps your passphrase out of your settings file, but requires that GPG agent be available and loaded with your private key. This is normal when running interactively. When running batch, say from *cron*, you can use the Linux *keychain* command to retain your GPG credentials for you.

avendesora_field

Specifies the name of the field in *Avendesora* that holds the encryption passcode. It is used along with *avendesora_account*. This setting is not needed if the field name is *Avendesora's* default.

borg_executable

The path to the *Borg* executable or the name of the *Borg* executable. By default it is simply `borg`.

check_after_create

Whether the archive or repository should be checked after an archive is created. May be one of the following: *False*, *True*, `"latest"`, `"all"`, or `"all in repository"`. If *False*, no checking is performed. If `"latest"`, only the archive just created is checked. If *True* or `"all"`, all archives associated with the current configuration are checked. Finally, if `"all in repository"`, all the archives contained in the repository are checked, including those associated with other archives. In all cases checks are performed on the repository and the archive or archives selected, but in none of the cases is data integrity verification performed. To check the integrity of the data you must explicitly run the *check command*. Regardless, the checking can be quite slow if `"all"` or `"all in repository"` are used.

colorscheme

A few commands colorize the text to convey extra information. You can optimize the tints of those colors to make them more visible and attractive. *colorscheme* should be set to "none", "light", or "dark". With "none" the text is not colored. In general it is best to use the "light" colorscheme on dark backgrounds and the "dark" colorscheme on light backgrounds.

compact_after_delete

If True, the *compact command* is run after deleting an archive or pruning a repository.

Note: This is an important setting if you are using *Borg 1.2* or later. You should either set this true or manage the compaction in another way. Setting it true results in slightly slower backups. The alternative is generally to configure *cron* or *anacron* to run the *compact* command routinely for you.

Do not use this setting if you are not using *Borg* version 1.2 or later.

---

configurations

The list of available *Emborg* configurations. To be usable the name of a configuration must be in this list and there must be a file of the same name in the `~/.config/emborg` directory.

The value may be specified as a list of strings or just as a string. If specified as a string, it is split on white space to form the list.

cronhub_url

This setting specifies the URL to use for cronhub.io. Normally it is not needed. If not specified `https://cronhub.io` is used. You only need to specify the URL in special cases.

cronhub_uuid

If this setting is provided, *Emborg* notifies cronhub.io when the archive is being created and whether the creation was successful. The value of the setting should be a UUID (a 32 digit hexadecimal number that contains 4 dashes). If given, this setting should be specified on an individual configuration. For example:

```
cronhub_uuid = '51cb35d8-2975-110b-67a7-11b65d432027'
```

default_configuration

The name of the configuration to use if one is not specified on the command line.

default_mount_point

The path to a directory that should be used if one is not specified on the *mount command* or *umount command* commands. When set the mount point directory becomes optional on these commands. You should choose a directory that itself is not subject to being backed up to avoid creating a loop. For example, you might consider something in /tmp:

```
default_mount_point = '/tmp/emborg'
```

do_not_expand

All settings that are specified as strings or lists of strings may contain placeholders that are expanded before use. The placeholder is replaced by the value it names. For example, in:

```
archive = '{host_name}-{{now}}'
```

host_name is a placeholder that is replaced by the host name of your computer before it is used (*now* is escaped using double braces and so does not act as a placeholder for *Emborg*.

do_not_expand is a list of names for settings that should not undergo placeholder replacement. The value may be specified as a list of strings or just as a string. If specified as a string, it is split on white space to form the list.

encoding

The encoding used when communicating with Borg. The default is utf-8, which is generally suitable for Linux systems.

encryption

The encryption mode that is used when first creating the repository. Common values are `none`, `authenticated`, `repokey`, and `keyfile`. The repository is encrypted if you choose `repokey` or `keyfile`. In either case the passphrase you provide does not encrypt repository. Rather the repository is encrypted using a key that is randomly generated by *Borg*. You passphrase encrypts the key. Thus, to restore your files you will need both the key and the passphrase. With `repokey` your key is copied to the repository, so `repokey` should only be used with trusted repositories. Use `keyfile` if the remote repository is not trusted. It does not copy the key to the repository, meaning that it is extremely important for you export the key using 'borg key export' and keep a copy in a safe place along with the passphrase.

Once encrypted, a passphrase is needed to access the repository. There are a variety of ways to provide it. *Borg* itself uses the *BORG_PASSPHRASE*, *BORG_PASSPHRASE_FD*, and *BORG_COMMAND* environment variables if set. *BORG_PASSPHRASE* contains the passphrase, or *BORG_PASSPHRASE_FD* is a file descriptor that provides the passphrase, or *BORG_COMMAND* contains a command that generates the passphrase. If none of those are set, *Emborg* looks to its own settings. If either *passphrase* or *passcommand* are set, they are used. If neither are set, *Emborg* uses *avendesora_account* if set. Otherwise no passphrase is available and the command fails if the repository is encrypted.

excludes

A list of files or directories to exclude from the backups. Typical value might be:

```
excludes = """
    ~/tmp
    ~/.local
    ~/.cache
    ~/.mozilla
    ~/.thunderbird
    ~/.config/google-chrome*
    ~/.config/libreoffice
    ~/**/__pycache__
    ~/**/*.pyc
    ~/**/.*.swp
    ~/**/.*.swo
"""
```

The value can either be specified as a list of strings or as a multi-line string with one exclude per line.

Emborg supports the same exclude patterns that Borg itself supports.

When specifying paths to excludes, the paths may be relative or absolute. When relative, they are taken to be relative to *working_dir*.

exclude_from

An alternative to *excludes*. You can list your excludes in one or more files, one per line, and then specify the file or files using the *exclude_from* setting:

```
exclude_from = '{config_dir}/excludes'
```

The value of *exclude_from* may either be a multi-line string, one file per line, or a list of strings. The string or strings would be the paths to the file or files that contain the list of files or directories to exclude. If given as relative paths, they are relative to *working_dir*. These files are processed directly by *Borg*, which does not allow ~ to represent users' home directories, unlike the patterns specified using *patterns*.

healthchecks_url

This setting specifies the URL to use for healthchecks.io. Normally it is not needed. If not specified `https://.hc-ping.com` is used. You only need to specify the URL in special cases.

healthchecks_uuid

If this setting is provided, *Emborg* notifies healthchecks.io when the archive is being created and whether the creation was successful. The value of the setting should be a UUID (a 32 digit hexadecimal number that contains 4 dashes). If given, this setting should be specified on an individual configuration. For example:

```
healthchecks_uuid = '51cb35d8-2975-110b-67a7-11b65d432027'
```

include

Can be a string or a list of strings. Each string specifies a path to a file. The contents of that file are read into *Emborg*. If the path is relative, it is relative to the file that includes it.

manage_diffs_cmd

Command to use to perform interactive file and directory comparisons using the `--interactive` option to the *compare command*. The command may be specified in the form of a string or a list of strings. If a string, it may contain the literal text `{archive_path}` and `{local_path}`, which are replaced by the two files or directories to be compared. If not, then the paths are simply appended to the end of the command as specified. Suitable commands for use in this setting include Vim with the DirDiff plugin, Meld, and presumably others such as *DiffMerge*, *Kompare*, *Diffuse*, *KDiff3*, etc. If you are a *Vim* user, another alternative is vdiff, which provides a more streamlined interface to *Vim/DirDiff*. Here are examples on how to configure *Vim*, *Meld* and *VDiff*:

```
manage_diffs_cmd = "meld"
manage_diffs_cmd = ["meld", "-a"]
manage_diffs_cmd = "gvim -f -c 'DirDiff {archive_path} {local_path}'"
manage_diffs_cmd = "vdiff -g"
```

The *compare command* mounts the remote archive, runs the specified command and then immediately unmounts the archive. As such, it is important that the command run in the foreground. By default, *gvim* runs in the background. You can tell this because if run directly in a shell, the shell immediately accepts new commands even though *gvim* is still active. To avoid this, the `-f` option is added to the *gvim* command line to indicate it should run in the foreground. Without this, you will see an error from *fusermount* indicating 'Device or resource busy'. If you get this message, you will have to close the editor and manually un-mount the archive.

manifest_default_format

A string that specifies the name of the default format. The name must be a key in *manifest_formats*. If not specified, `short` is used.

manifest_formats

A dictionary that defines how the output of the manifest command is to be formatted. The default value for *manifest_formats* is:

```
manifest_formats = dict(
    name = "{path}",
    short = "{path}{Type}",
    date = "{mtime} {path}{Type}",
    size = "{size:8} {path}{Type}",
    si = "{Size:6.2} {path}{Type}",
    owner = "{user:8} {path}{Type}",
    group = "{group:8} {path}{Type}",
    long = '{mode:10} {user:6} {group:6} {size:8} {mtime} {path}{extra}',
)
manifest_default_format = 'short'
```

Notice that 8 formats are defined:

name
: used when `--name-only` is specified.

short
: used by when `--short` is specified and when sorting by name.

date
: used by default when sorting by date.

size
: size in bytes (fixed format).

si
: size in bytes (SI format), used by default when sorting by size.

owner
: used by default when sorting by owner.

group
: used by default when sorting by group.

long
: used when `--long` is specified.

Your *manifest_formats* need not define all or even any of these formats. The above example shows the formats that are predefined in *Emborg*. You do not need to specify them again. Anything you specify will override the predefined versions, and you can add additional formats.

The formats may contain the fields supported by the Borg list command. In addition, Emborg provides some variants:

***MTime*, *CTime*, *ATime*:**
: The *Borg mtime*, *ctime*, and *atime* fields are simple strings, these variants are Arrow objects that support formatting options. For example:

```
date = "{MTime:ddd YYYY-MM-DD HH:mm:ss} {path}{Type}",
```

***Size*, *CSize*, *DSize*, *DCSize*:**
The *Borg size*, *csize*, *dsize* and *dctime* fields are simple integers, these variants are QuantiPhy objects that support formatting options. For example:

```
size = "{Size:5.2r} {path}{Type}",
size = "{Size:7.2b} {path}{Type}",
```

***Type*:**
Displays / for directories, @ for symbolic links, and | for named pipes.

QuantiPhy objects allow you to format the size using SI scale factors (K, Ki, M, Mi, etc.). *Arrow* objects allow you to format the date and time in a wide variety of ways. Any use of *QuantiPhy* or *Arrow* can slow long listings considerably.

The fields support Python format strings, which allows you to specify how they are to be formatted. Anything outside a field is copied literally.

must_exist

Specify paths to files that must exist before *create command* can be run. This is used to assure that relevant file systems are mounted before making backups of their files.

May be specified as a list of strings or as a multi-line string with one path per line.

needs_ssh_agent

A Boolean. If true, *Emborg* will issue an error message and refuse to run if an SSH agent is not available.

notifier

A string that specifies the command used to interactively notify the user of an issue. A typical value is:

```
notifier = 'notify-send -u critical {prog_name} "{msg}"'
```

Any of the following names may be embedded in braces and included in the string. They will be replaced by their value:

msg: The message for the user.
hostname: The host name of the system that *Emborg* is running on.
user_name: The user name of the person that started *Emborg*
prog_name: The name of the *Emborg* program.

The notifier is only used if the command is not running from a TTY.

Use of *notifier* requires that you have a notification daemon installed (ex: Dunst). The notification daemon provides the *notify-send* command. If you do not have the *notify-send* command, do not set *notifier*.

The *notify* and *notifier* settings operate independently. You may specify none, one, or both. Generally, one uses just one: *notifier* if you primarily use *Emborg* interactively and *notify* if used from cron or anacron.

notify

A string that contains one or more email addresses separated with spaces. If specified, an email will be sent to each of the addresses to notify them of any problems that occurred while running *Emborg*.

The email is only sent if the command is not running from a TTY.

Use of *notify* requires that you have a mail daemon installed (ex: PostFix configured as a null client). The mail daemon provides the *mail* command. If you do not have the *mail* command, do not set *notify*.

The *notify* and *notifier* settings operate independently. You may specify none, one, or both. Generally, one uses just one: *notifier* if you primarily use *Emborg* interactively and *notify* if used from cron or anacron.

passcommand

A string that specifies a command to be run by *BORG* to determine the pass phrase for the encryption key. The standard out of this command is used as the pass phrase. This string is passed to *Borg*, which executes the command.

Here is an example of a passcommand that you can use if your GPG agent is available when *Emborg* is run. This works if you are running it interactively, or in a cron script if you are using keychain to provide you access to your GPG agent:

```
passcommand = 'gpg -qd /home/user/.store-auth.gpg'
```

This is used as an alternative to *passphrase* when it is desirable to keep the passphrase out of your configuration file.

passphrase

A string that specifies the pass phrase for the encryption key. This string is passed to *Borg*. When specifying a pass phrase you should be careful to assure that the configuration file that contains is only readable by the user and nobody else.

prune_after_create

A Boolean. If true the *prune command* is run after creating an archive.

report_diffs_cmd

Command used to perform file and directory comparisons using the *compare command*. The command may be specified in the form of a string or a list of strings. If a string, it may contain the literal text `{archive_path}` and `{local_path}`, which are replaced by the two files or directories to be compared. If not, then the paths are simply appended to the end of the command as specified. Suitable commands for use in this setting include `diff -r` the and `colordiff -r`. Here are examples of two different but equivalent ways of configuring *diff*:

```
report_diffs_cmd = "diff -r"
report_diffs_cmd = "diff -r {archive_path} {local_path}"
```

You may prefer to use *colordiff*, which is like *diff* but in color:

```
report_diffs_cmd = "colordiff -r"
```

repository

The destination for the backups. A typical value might be:

```
repository = 'archives:/mnt/backups/{host_name}-{user_name}-{config_name}'
```

where in this example 'archives' is the hostname and /mnt/backups is the absolute path to the directory that is to contain your Borg repositories, and {host_name}-{user_name}-{config_name} is the directory to contain this repository. For a local repository you would use something like this:

```
repository = '/mnt/backups/{host_name}-{user_name}-{config_name}'
```

These examples assume that */mnt/backups* contains many independent repositories, and that each repository contains the files associated with a single backup configuration. Borg allows you to make a repository the target of more than one backup configuration, and in this way you can further benefit from its ability to de-duplicate files. In this case you might want to use a less granular name for your repository. For example, a particular user could use a single repository for all their configurations on all their hosts using:

```
repository = '/mnt/backups/{user_name}'
```

When more than one configuration shares a repository you should specify the *glob_archives* setting so that each configuration can recognize its own archives.

A local repository should be specified with an absolute path, and that path should not contain a colon (`:`) to avoid confusing the algorithm that determines whether the repository is local or remote.

run_after_backup, run_after_last_backup

Specifies commands that are to be run after the *create* command successfully completes. These commands often recreate useful files that were deleted by the *run_before_backup* commands.

May be specified as a list of strings or as a multi-line string with one command per line (lines that begin with # are ignored).

The commands specified in *run_after_backup* are run each time an archive is created whereas commands specified in *run_after_last_backup* are run only if the configuration is run individually or if it is the last run in a composite configuration. For example, imagine a composite configuration *home* that consists of two children, *local* and *remote*, and imagine that both are configured to run the command *restore* after they are run. If *run_after_backup* is used to specify *restore*, then running `emborg -c home create` results in *restore* being run twice, after both the *local* and *remote* archives are created. However, if *run_after_last_backup* is used, *restore* is only run once, after the *remote* archive is created. Generally, one specifies identical commands to *run_after_last_backup* for each configuration in a composite configuration with the intent that the commands will be run only once regardless whether the configurations are run individually or as a group.

For example, the following runs *Borg-Space* after each back-up to record the size history of your repository:

```
run_after_backup = [
    'borg-space -r -m "Repository is now {{size:.2}}." {config_name}'
]
```

run_before_backup, run_before_first_backup

Specifies commands that are to be run before the *create* command starts the backup. These commands often delete large files that can be easily recreated from those files that are backed up.

May be specified as a list of strings or as a multi-line string with one command per line (lines that begin with # are ignored).

The commands specified in *run_before_backup* are run each time an archive is created whereas commands specified in *run_before_first_backup* are run only if the configuration is run individually or if it is the first run in a composite configuration. For example, imagine a composite configuration *home* that consists of two children, *local* and *remote*, and imagine that both are configured to run the command *clean* before they are run. If *run_before_backup* is used to specify *clean*, then running `emborg -c home create` results in *clean* being run twice, before both the *local* and *remote* archives are created. However, if *run_before_first_backup* is used, *clean* is only run once, before the *local* archive is created. Generally, one specifies identical commands to *run_before_first_backup* for each configuration in a composite configuration with the intent that the commands will be run only once regardless whether the configurations are run individually or as a group.

run_before_borg, run_after_borg

Specifies commands that are to be run before the first *Borg* command is run or after the last one is run. These can be used, for example, to mount and then unmount a remote repository, if such a thing is needed.

May be specified as a list of strings or as a multi-line string with one command per line (lines that begin with # are ignored).

show_progress

Show progress when running *Borg*'s *create* command. You also get this by adding the `--progress` command line option to the *create* command, but if this option is set True then this command will always show the progress.

show_stats

Show statistics when running *Borg*'s *create*, *delete* and *prune* commands. You can always get this by adding the `--stats` command line option to the appropriate commands, but if this option is set True then these commands will always show the statistics. If the statistics are not requested, they will be recorded in the log file rather than being displayed.

Statistics are incompatible with the –dry-run option and will be suppressed on trial runs.

src_dirs

A list of strings, each of which specifies a directory to be backed up. May be specified as a list of strings or as a multi-line string with one source directory per line.

When specifying the paths to the source directories, the paths may be relative or absolute. When relative, they are taken to be relative to *working_dir*.

ssh_command

A string that contains the command to be used for SSH. The default is `"ssh"`. This can be used to specify SSH options.

verbose

A Boolean. If true *Borg* is run in verbose mode and the output from *Borg* is output by *Emborg*.

10.3.10 Borg Settings

These settings control the behavior of *Borg*. Detailed descriptions can be found in the Borg documentation.

append_only

Create an append-only mode repository.

chunker_params

Parameters used by the chunker command. More information is available from chunker_params Borg documentation.

compression

The name of the desired compression algorithm.

exclude_caches

Exclude directories that contain a CACHEDIR.TAG file.

exclude_if_present

Exclude directories that are tagged by containing a filesystem object with the given NAME

exclude_nodump

Exclude files flagged NODUMP.

glob_archives

A glob string that a backup configuration uses to recognize its archives when more than one configuration is sharing the same repository. A glob string is a string that is expected to match the name of the archives. It must contain at least one asterisk (`*`). Each asterisk will match any number of contiguous characters. For example, a *glob_archives* setting of `home-*` will match `home-2022-10-23T19:11:04`.

glob_archives is required if you save the archives of multiple backup configurations to the same repository. Otherwise it is not needed. It is used by the *Check*, *Delete*, *Info*, *List*, *Mount*, and *Prune* commands to filter out archives not associated with the desired backup configuration.

lock_wait

Wait at most SECONDS for acquiring a repository/cache lock (default: 1)

keep_within

Keep all archives within this time interval.

keep_last

Number of the most recent archives to keep.

keep_minutely

Number of minutely archives to keep.

keep_hourly

Number of hourly archives to keep.

keep_daily

Number of daily archives to keep.

keep_weekly

Number of weekly archives to keep.

keep_monthly

Number of monthly archives to keep.

keep_yearly

Number of yearly archives to keep.

one_file_system

Stay in the same file system and do not store mount points of other file systems.

patterns

A list of files or directories to exclude from the backups. Typical value might be:

```
patterns = """
    R /
    - /home/*/.cache
    - /home/*/Downloads

    # include susan's home
    + /home/susan

    # don't backup the other home directories
    - /home/*
"""
```

The value can either be specified as a list of strings or as a multi-line string with one pattern per line.

Patterns are a new experimental feature of *Borg*. They allow you to specify what to back up and what not to in a manner that is more flexible than *src_dirs* and *excludes* allows, and can fully replace them.

For example, notice that /home/susan is included while excluding the directory that contains it (/home).

Emborg supports the same patterns that Borg itself supports.

When specifying paths in patterns, the paths may be relative or absolute. When relative, they are taken to be relative to *working_dir*.

patterns_from

An alternative to *patterns*. You can list your patterns in one or more files, one per line, and then specify the file or files using the *exclude_from* setting.

```
patterns_from = '{config_dir}/patterns'
```

The value of *patterns_from* may either be a multi-line string, one file per line, or a list of strings. The string or strings would be the paths to the file or files that contain the patterns. If given as relative paths, they are relative to *working_dir*. These files are processed directly by *Borg*, which does not allow ~ to represent users' home directories, unlike the patterns specified using *patterns*.

prefix

Only consider archive names starting with this prefix.

As of Borg 1.2 *prefix* is deprecated and should no longer be used. Use *glob_archives* instead. It provides the same basic functionality in a way that is a little more general. For more information, see *archive*.

Prior to the deprecation of *prefix* it was common in *Emborg* settings file to just specify *prefix* and not specify *archive* with the understanding that the default value of *archive* is `{prefix}-{{now}}`. So you might have something like:

```
prefix = '{config_name}-'
```

in your settings file. This can be converted to:

```
archive = '{config_name}-{{now}}'
glob_archives = '{config_name}-*'
```

without changing the intent.

remote_path

Name of *Borg* executable on remote platform.

sparse

Detect sparse holes in input (supported only by fixed chunker).

Requires *Borg* version 1.2 or newer.

threshold

Sets minimum threshold for saved space when compacting a repository with the *compact command*. Value is given in percent.

Requires *Borg* version 1.2 or newer.

remote_ratelimit

Set remote network upload rate limit in KiB/s (default: 0=unlimited).

Borg has deprecated *remote_ratelimit* in version 1.2. If you are seeing this warning, you should rename *remote_ratelimit* to *upload_ratelimit* in your *Emborg* settings file.

umask

Set umask. This is passed to *Borg*. It uses it when creating files, either local or remote. The default is 0o077.

upload_buffer

Set network upload buffer size in MiB. By default no buffer is used. Requires *Borg* version 1.2 or newer.

upload_ratelimit

Set upload rate limit in KiB/s when writing to a remote network (default: 0=unlimited).

Use *upload_ratelimit* when using *Borg* version 1.2 or higher, otherwise use *remote_ratelimit*.

working_dir

All relative paths specified in the configuration files (other than those specified to *include*) are relative to *working_dir*.

Emborg changes to the working directory before running the *Borg create* command, meaning that relative paths specified as roots, excludes, or patterns (*src_dirs*, *excludes*, *patterns*, *exclude_from* or *patterns_from*) are taken to be relative to the working directory. If you use absolute paths for your roots, excludes, and pattern, then the working directory must be set to `/`.

To avoid confusion, it is recommended that all other paths in your configuration be specified using absolute paths (ex: *default_mount_point*, *must_exist*, *patterns_from*, and *exclude_from*).

If specified, *working_dir* must be specified using an absolute path. If not specified, *working_dir* defaults to /.

10.4 Monitoring

10.4.1 Overdue

Checking for Overdue Backups from the Server

Emborg contains an additional executable, *emborg-overdue*, that can be run on the destination server to determine whether the backups have been performed recently. It reads its own settings file in ~/.config/emborg/overdue.conf that is also a Python file and may contain the following settings:

```
default_maintainer (email address -- mail is sent to this person upon failure)
default_max_age (hours)
dumper (email address -- mail is sent from this person)
root (default directory for repositories)
repositories (string or array of dictionaries)
```

Here is an example config file:

```
default_maintainer = 'root@continuum.com'
dumper = 'dumper@continuum.com'
default_max_age = 12  # hours
root = '/mnt/borg-backups/repositories'
repositories = [
    dict(host='mercury (/)', path='mercury-root-root'),
    dict(host='venus (/)', path='venus-root-root'),
    dict(host='earth (/)', path='earth-root-root'),
    dict(host='mars (/)', path='mars-root-root'),
    dict(host='jupiter (/)', path='jupiter-root-root'),
    dict(host='saturn (/)', path='saturn-root-root'),
    dict(host='uranus (/)', path='uranus-root-root'),
    dict(host='neptune (/)', path='neptune-root-root'),
    dict(host='pluto (/)', path='pluto-root-root'),
]
```

The dictionaries in *repositories* can contain the following fields: *host*, *path*, *maintainer*, *max_age*. *host* is an arbitrary string that is used as description of the repository. It is included in the email that is sent when problems occur to identify the backup and so should be unique. It is a good idea for it to contain both the host name and the source directory being backed up. *path* is either the archive name or a full absolute path to the archive. If *path* is an absolute path, it is used, otherwise it is added to the end of *root*. *maintainer* is an email address, an email is sent to this address if there is an issue. *max_age* is the number of hours that may pass before an archive is considered overdue.

repositories can also be specified as multi-line string:

```
repositories = """
    # HOST      | NAME or PATH      | MAINTAINER          | MAXIMUM AGE (hours)
    mercury (/) | mercury-root-root |                     |
    venus (/)   | venus-root-root   |                     |
    earth (/)   | earth-root-root   |                     |
    mars (/)    | mars-root-root    |                     |
    jupiter (/) | jupiter-root-root |                     |
```

(continues on next page)

(continued from previous page)

```
    saturn (/)  | saturn-root-root  |                       |
    uranus (/)  | uranus-root-root  |                       |
    neptune (/) | neptune-root-root |                       |
    pluto (/)   | pluto-root-root   |                       |
"""
```

If *repositories* is a string, it is first split on newlines, anything beyond a # is considered a comment and is ignored, and the finally the lines are split on '|' and the 4 values are expected to be given in order. If the *maintainer* is not given, the *default_maintainer* is used. If *max_age* is not given, the *default_max_age* is used.

To run the program interactively, just make sure *emborg-overdue* has been installed and is on your path. Then type:

```
$ emborg-overdue
```

It is also common to run *emborg-overdue* on a fixed schedule from cron. To do so, run:

```
$ crontab -e
```

and add something like the following:

```
34 5 * * * ~/.local/bin/emborg-overdue --mail > ~/.local/share/emborg/emborg-overdue.out␣
↪2>&
```

or:

```
34 5 * * * ~/.local/bin/emborg-overdue --quiet --mail
```

to your crontab.

The first example runs emborg-overdue at 5:34 AM every day while saving the output into a file. The use of the `--mail` option causes *emborg-overdue* to send mail to the maintainer when backups are found to be overdue.

Note: By default Linux machines are not configured to send email. If you are using the `--mail` option to *emborg-overdue* be sure that to check that it is working. You can do so by sending mail to your self using the *mail* command. If you do not receive your test message you will need to set up email forwarding on your machine. You can do so by installing and configuring PostFix as a null client.

The second example is similar except the output is suppressed rather than being saved to a file.

Alternately you can run *emborg-overdue* from cron.daily (described in the *root example*).

Checking for Overdue Backups from the Client

emborg-overdue can also be configured to run on the client. This can be used when you do not control the server and so cannot run *emborg-overdue* there. The configuration is identical, except you give the path to the *latest.nt* file. For example:

```
default_maintainer = 'me@continuum.com'
dumper = 'me@continuum.com'
default_max_age = 12  # hours
root = '~/.local/share/emborg'
repositories = [
    dict(host='earth (cache)', path='cache.latest.nt', max_age=0.2),
```

(continues on next page)

(continued from previous page)

```
    dict(host='earth (home)', path='home.latest.nt'),
]
```

Again, *emborg-overdue* is generally run from cron.

10.4.2 Monitoring Services

Various monitoring services are available on the web. You can configure *Emborg* to notify them when back-up jobs have started and finished. These services allow you to monitor many of your routine tasks and assure they have completed recently and successfully.

There are many such services available and they are not difficult to add. If the service you prefer is not currently available, feel free to request it on Github or add it yourself and issue a pull request.

CronHub.io

When you sign up with cronhub.io and configure the health check for your *Emborg* configuration, you will be given a UUID (a 32 digit hexadecimal number partitioned into 5 parts by dashes). Add that to the following setting in your configuration file:

```
cronhub_uuid = '51cb35d8-2975-110b-67a7-11b65d432027'
```

If given, this setting should be specified on an individual configuration. It causes a report to be sent to *CronHub* each time an archive is created. A successful report is given if *Borg* returns with an exit status of 0 or 1, which implies that the command completed as expected, though there might have been issues with individual files or directories. If *Borg* returns with an exit status of 2 or greater, a failure is reported.

HealthChecks.io

When you sign up with healthchecks.io and configure the health check for your *Emborg* configuration, you will be given a UUID (a 32 digit hexadecimal number partitioned into 5 parts by dashes). Add that to the following setting in your configuration file:

```
healthchecks_uuid = '51cb35d8-2975-110b-67a7-11b65d432027'
```

If given, this setting should be specified on an individual configuration. It causes a report to be sent to *HealthChecks* each time an archive is created. A successful report is given if *Borg* returns with an exit status of 0 or 1, which implies that the command completed as expected, though there might have been issues with individual files or directories. If *Borg* returns with an exit status of 2 or greater, a failure is reported.

10.5 Accessories

10.5.1 Borg-Space

Borg-Space is a utility that reports and tracks the space required by your *Borg* repositories. It also allows you to graph the space used over time.

The following is an example of a graph generated by *borg-space* that allowed me to catch a problem that resulted in excessive growth in in the space required to hold my repository: in the switch from *Borg 1.1* to *Borg 1.2*, I had neglected to implement a compaction strategy. The problem was resolved on April 5th.

10.5.2 Logging with ntLog

ntLog is a log file aggregation utility.

When run *Emborg* writes over a previously generated logfile. This becomes problematic if you have one cron script that runs *create* frequently and another that runs a command like *prune* less frequently. If there is trouble with the *prune* command it will be difficult to see and resolve because its logfile will be overwritten by subsequent *create* commands.

ntlog can be run after each *Emborg* run to aggregate the individual logfile from each run into a single accumulating log file. To arrange this you can use *run_after_borg*:

```
run_after_borg = 'ntlog --keep-for 7d ~/.local/share/emborg/{config_name}.log'
```

This accumulates the log files as they are created to ~/.local/share/emborg/{config_name}.log.nt.

10.6 Examples

When first run, *Emborg* creates the settings directory and populates it with two configurations that you can use as starting points. Those two configurations make up our first two examples.

10.6.1 Root

The *root* configuration is a suitable starting point for someone that wants to backup an entire machine, including both system and user files. In order to have permission to access the files, one must run this configuration as the *root* user.

This configuration was constructed assuming that the backups would be run automatically at a fixed time using cron. Since this user only has one configuration, it is largely arbitrary which file each setting resides in, however both files must exist, and the *settings* file must contain *configurations* and *default_configuration*.

Here is the contents of the settings file: /root/.config/emborg/settings:

```
configurations = 'root'
default_configuration = 'root'

# basic settings
notify = "root@continuum.com"
upload_ratelimit = 2000     # bandwidth limit in kbps
prune_after_create = True
check_after_create = 'latest'

# repository settings
repository = 'backups:/mnt/backups/{host_name}-{user_name}-{config_name}'
archive = '{prefix}{{now:%Y%m%d}}'
prefix = '{config_name}-'
compression = 'lz4'

# shared filter settings
exclude_if_present = '.nobackup'
exclude_caches = True
```

(continues on next page)

(continued from previous page)

```
# prune settings
keep_within = '1d'          # keep all archives created within this interval
keep_hourly = 48            # number of hourly archives to keep
keep_daily = 14             # number of daily archives to keep
keep_weekly = 8             # number of weekly archives to keep
keep_monthly = 24           # number of monthly archives to keep
keep_yearly = 24            # number of yearly archives to keep
```

In this case we are assuming that *backups* (used in *repository*) is an entry in your SSH config file that points to the server that stores your repository. To be able to run this configuration autonomously from cron, *backups* must be configured to use a private key that does not have a passphrase.

And here is the contents of the *root* configuration file: /root/.config/emborg/root:

```
# Settings for root configuration
passphrase = 'carvery overhang vignette platitude pantheon sissy toddler truckle'
encryption = 'repokey'
one_file_system = False

src_dirs = '/'
excludes = '''
    /dev
    /home/*/.cache
    /mnt
    /proc
    /run
    /sys
    /tmp
    /var/cache
    /var/lock
    /var/run
    /var/tmp
'''                         # list of files or directories to skip
```

This file contains the passphrase, and so you should be careful to set its permissions so that nobody but root can see its contents. Also, this configuration uses *repokey* as the encryption method, which is suitable when you control the server that holds the repository and you know it to be secure.

Once this configuration is complete and has been tested, you would want to add a crontab entry so that it runs on a routine schedule. On servers that are always running, you could use *crontab -e* and add an entry like this:

```
30 03 * * * emborg --mute --config root create
```

For individual workstations or laptops that are likely to be turned off at night, one would instead create an executable script in /etc/cron.daily that contains the following:

```
#/bin/sh
# Run root backups

emborg --mute --config root create
```

Assume that this file is named *emborg*. Then after creating it, you would make it executable with:

```
$ chmod a+x /etc/cron.daily/emborg
```

Scripts in /etc/cron.daily are one once a day, either at a fixed time generally early in the morning or, if not powered up at that time, shortly after being powered up.

10.6.2 User

The *home* configuration is a suitable starting point for someone that just wants to backup their home directory on their laptop. In this example, two configurations are created, one to be run manually that copies all files to a remote repository, and a second that runs every few minutes and creates snapshots of key working directories. This second allows you to quickly recover from mistakes you make during the day without having to go back to yesterday's copy of a file as a starting point.

Here is the contents of the shared settings file: ~/.config/emborg/settings.

```
# configurations
configurations = 'home snapshots'
default_configuration = 'home'

# basic settings
notifier = 'notify-send -u normal {prog_name} "{msg}"'

# repository settings
compression = 'lz4'

# shared filter settings
exclude_if_present = '.nobackup'
exclude_caches = True
```

Home

Here is the contents of the *home* configuration file: ~/.config/emborg/home. This configuration backs up to a remote untrusted repository and is expected to be run interactively, perhaps once per day.

```
repository = 'backups:/mnt/borg-backups/repositories/{host_name}-{user_name}-{config_
↪name}'
prefix = '{config_name}-'
encryption = 'keyfile'
avendesora_account = 'laptop-borg'
needs_ssh_agent = True
upload_ratelimit = 2000
prune_after_create = True
check_after_create = 'latest'

src_dirs = '~'                      # paths to be backed up
excludes = '''
    ~/.cache
    **/.hg
    **/.git
    **/__pycache__
    **/*.pyc
```

(continues on next page)

(continued from previous page)

```
    **/.*.swp
    **/.*.swo
    **/*~
'''

run_before_backup = '(cd ~/src; ./clean)'

# prune settings
keep_within = '1d'                      # keep all archives created within this␣
↪interval
keep_hourly = 48                        # number of hourly archives to keep
keep_daily = 14                         # number of daily archives to keep
keep_weekly = 8                         # number of weekly archives to keep
keep_monthly = 24                       # number of monthly archives to keep
keep_yearly = 24                        # number of yearly archives to keep
```

In this case we are assuming that *backups* (used in *repository*) is an entry in your SSH config file that points to the server that stores your repository. *backups* should be configured to use a private key and that key should be preloaded into your SSH agent.

This passphrase for this configuration is kept in Avendesora, and the encryption method is *keyfile*. As such, it is critical that you extract the keyfile from *Borg* and copy it and your *Avendesora* files to a safe place so that both the keyfile and passphrase are available if you lose your disk. You can use SpareKeys to do this for you. Otherwise extract the keyfile using:

```
$ emborg borg key export @repo key.borg
```

cron is not used for this configuration because the machine, being a laptop, is not guaranteed to be on at any particular time of the day. So instead, you would simply run *Emborg* on your own at a convenient time using:

```
$ emborg
```

You can use the *Emborg due* command to remind you if a backup is overdue. You can wire it into status bar programs, such as *i3status* to give you a visual reminder, or you can configure cron to check every hour and notify you if they are overdue. This one triggers a notification:

```
0 * * * * emborg --mute due --days 1 || notify-send 'Backups are overdue'
```

And this one sends an email:

```
0 * * * * emborg --mute due --days 1 --mail me@mydomain.com
```

Alternately, you can use *emborg-overdue*.

Snap Shots

And finally, here is the contents of the *snapshots* configuration file: ~/.config/emborg/snapshots.

```
repository = '~/.cache/snapshots'
encryption = 'none'

src_dirs = '~'
excludes = '''
    ~/.cache
    ~/media
    **/.hg
    **/.git
    **/__pycache__
    **/*.pyc
    **/.*.swp
    **/.*.swo
    **/.~
'''

# prune settings
keep_hourly = 12
prune_after_create = True
check_after_create = False
```

To run this configuration every 10 minutes, add the following entry to your crontab file using 'crontab -e':

```
0,10,20,30,40,50 * * * * emborg --mute --config snapshots create
```

10.6.3 Rsync.net

Rsync.net is a commercial option for off-site storage. In fact, they give you a discount if you use Borg Backup.

Once you sign up for *Rsync.net* you can access your storage using *sftp*, *scp*, *rsync* or *borg* of course. *ssh* access is also available, but only for a limited set of commands.

You would configure *Emborg* for *Rsync.net* in much the same way you would for any remote server. Of course, you should use some form of *keyfile* based encryption to keep your files secure. The only thing to be aware of is that by default they provide a old version of borg. To use a newer version, set the `remote_path` to `borg1`.

```
repository = '78548@ch-s012.rsync.net:repo'
encryption = 'keyfile'
remote_path = 'borg1'

...
```

In this example, `78548` is the user name and `ch-s012.rsync.net` is the server they assign to you. `repo` is the name of the directory that is to contain your *Borg* repository. You are free to name it whatever you like and you can have as many as you like, with the understanding that you are constrained in the total amount of storage you consume.

10.6.4 BorgBase

BorgBase is another commercial alternative for *Borg Backups*. It allows full *Borg* access, append-only *Borg* access, and *rsync* access, though each form of access requires its own unique SSH key.

Again, you should use some form of *keyfile* encryption to keep your files secure, and *BorgBase* recommends *Blake2* encryption as being the fastest alternative.

```
repository = 'zMNZCv4B@zMNZCv4B.repo.borgbase.com:repo'
encryption = 'keyfile-blake2'

...
```

In this example, `zMNZCv4B` is the user name and `zMNZCv4B.repo.borgbase.com` is the server they assign to you. You may request any number of repositories, with each repository getting its own username and hostname. `repo` is the name of the directory that is to contain your *Borg* repository and cannot be changed.

10.7 Python API

Emborg has a simple API that allows you to run borg commands. Here is an example taken from sparekeys that exports the keys from your *Borg* repository so then can be backed up separately:

```
from emborg import Emborg
from pathlib import Path

destination = Path('keys')

with Emborg('home') as emborg:
    borg = emborg.run_borg(
        cmd = 'key export',
        args = [emborg.destination(), destination / '.config/borg.repokey']
    )
    if borg.stdout:
        print(borg.stdout.rstrip())
```

Emborg takes the config name as an argument, if not given the default config is used. You can also pass list of *Emborg* options and the path to the configurations directory.

Emborg provides the following useful methods and attributes:

configs

The list of configs associated with the requested config. If a scalar config was requested, the list be a list with a single member, the requested config. If the requested config is a composite config, the list consists of all the member configs of the requested config.

repository

The path to the repository.

version

The *Emborg* version number as a 3-tuple (major, minor, patch).

destination(archive)

Returns the full path to the archive. If Archive is False or None, then the path to the repository it returned. If Archive is True, then the default archive name as taken from settings file is used. This is only appropriate when creating new repositories.

run_borg(cmd, args, borg_opts, emborg_opts)

Runs a *Borg* command.

cmd is the desired *Borg* command (ex: 'create', 'prune', etc.).

args contains the command line arguments (such as the repository or archive). It may also contain any additional command line options not automatically provided. It may be a list or a string. If it is a string, it is split at white space.

borg_opts are the command line options needed by *Borg*. If not given, it is created for you by *Emborg* based upon your configuration settings.

Finally, *emborg_opts* is a list that may contain any of the following options: 'verbose', 'narrate', 'dry-run', or 'no-log'.

This function runs the *Borg* command and returns a process object that allows you access to stdout via the *stdout* attribute.

run_borg_raw(args)

Runs a raw *Borg* command without interpretation except for replacing a `@repo` argument with the path to the repository.

args contains all command line options and arguments except the path to the executable.

borg_options(cmd, emborg_opts)

This function returns the default *Borg* command line options, those that would be used in *run_borg* if *borg_opts* is not set. It can be used when constructing a custom *borg_opts*.

value(name, default='')

Returns the value of a scalar setting from an *Emborg* configuration. If not set, *default* is returned.

values(name, default=())

Returns the value of a list setting from an *Emborg* configuration. If not set, *default* is returned.

Of these entry points, only *configs* works with composite configurations.

You can examine the emborg/command.py file for inspiration and examples on how to use the *Emborg* API.

10.7.1 Example

A command that queries one or more configs and prints the total size of its archives. This example is a simplified version of the *Emborg* accessory available from Borg-Space.

```
#!/usr/bin/env python3
"""
Borg Repository Size

Reports on the current size of one or more Borg repositories managed by Emborg.

Usage:
    borg-space [options] [<config>...]

Options:
    -m <msg>, --message <msg>   template to use when building output message
```

(continues on next page)

(continued from previous page)

```
<msg> may contain {size}, which is replaced with the measured size, and
{config}, which is replaced by the config name.
If no replacements are made, size is appended to the end of the message.
"""

import arrow
from docopt import docopt
from emborg import Emborg
from quantiphy import Quantity
from inform import Error, display
import json

now = str(arrow.now())

cmdline = docopt(__doc__)
show_size = not cmdline['--quiet']
record_size = cmdline['--record']
message = cmdline['--message']

try:
    requests = cmdline['<config>']
    if not requests:
        requests = ['']  # this gets the default config

    for request in requests:
        # expand composite configs
        with Emborg(request, emborg_opts=['no-log']) as emborg:
            configs = emborg.configs

        for config in configs:
            with Emborg(config, emborg_opts=['no-log']) as emborg:

                # get name of latest archive
                borg = emborg.run_borg(
                    cmd = 'list',
                    args = ['--json', emborg.destination()]
                )
                response = json.loads(borg.stdout)
                try:
                    archive = response['archives'][-1]['archive']
                except IndexError:
                    raise Error('no archives available.', culprit=config)

                # get size info for latest archive
                borg = emborg.run_borg(
                    cmd = 'info',
                    args = ['--json', emborg.destination(archive)]
                )
                response = json.loads(borg.stdout)
                size = response['cache']['stats']['unique_csize']

                # report the size
```

(continues on next page)

(continued from previous page)

```
                size_in_bytes = Quantity(size, 'B')
                if not message:
                    message = '{config}: {size}'
                msg = message.format(config=config, size=size_in_bytes)
                if msg == message:
                    msg = f'{message}: {size_in_bytes}'
                display(msg)

except Error as e:
    e.report()
```

10.8 Releases

10.8.1 Latest development release

Version: 1.38
Released: 2023-11-04

10.8.2 1.38 (2023-11-04)

- Added 'last checked date' reporting to *due command*.
- Do not run *check –repair* and *compact* commands if *–dry-run* is requested.
- Pass output of *Borg create* command to hooks to allow it to be reported to healthchecks.io.

10.8.3 1.37 (2023-05-18)

- Add missing dependency.

10.8.4 1.36 (2023-05-15)

This release provides new mechanisms that allow you to monitor your pruning and compaction operations to help assure that these activities are not neglected. Both a prune and a compact operation must be performed to release disk space by eliminating expired archives. The combination of these to operations is referred to by *Emborg* as a squeeze.

- specifying an integer for `--date` now finds archive by index.
- *due* and *info* commands now report the latest *prune* and *compact* operations as well as the latest *create* operation.

Note: If you use *emborg-overdue* from the client you will need to change the paths you specify in *overdue.conf*. They now need to end in `.latest.nt` rather than `.lastbackup`.

Note: If you use *Borg-Space*, you will need to upgrade to version 2.

10.8.5 1.35 (2023-03-20)

- Improved the time resolution in *due* command.
- Added *si* format to *manifest* command.
- Allow *config_dir* to be specified through API.

10.8.6 1.34 (2022-11-03)

- Added ability to apply the *info* command to a particular archive.

10.8.7 1.33 (2022-10-22)

- Added *compare* command.
- Added *manage_diffs_cmd* and *report_diffs_cmd* settings.
- Allow ~/.config/emborg to always hold settings files if user prefers.

10.8.8 1.32 (2022-04-01)

- Fixed issues associated with *compact_after_delete* setting.

10.8.9 1.31 (2022-03-21)

- Enhanced *Emborg* to support new Borg 1.2 features.
  - Added *compact command*
  - Added *compact_after_delete*, *chunker_params*, *sparse*, *threshold*, *upload_ratelimit*, *upload_buffer* settings.
- Added the *run_before_borg and run_after_borg* settings.
- Added the `--cache-only` option and the ability to delete multiple archives at one time to the *delete command*.

10.8.10 1.30 (2022-01-04)

- Fix some issues with relative paths.

10.8.11 1.29 (2021-12-18)

- Do not signal failure to hooks if Borg completes normally, even if there were warnings.
- Return an exit status of 1 if *Emborg* runs to completion but with exceptions, and 2 if it cannot complete normally due to a error or errors.

10.8.12 1.28 (2021-11-06)

- Suppress log file generation for *configs*, *due*, *help*, *log*, *settings* and *version* commands.
- Add *version* to the API.

10.8.13 1.27 (2021-09-21)

- Improve the logging for composite configurations.
- Add support for Borg-Space, a utility that allows you to track and plot disk space usage for your *Borg* repositories over time.

10.8.14 1.26 (2021-09-03)

- Improve the tests.
- Allow access to names of child configs through API.

10.8.15 1.25 (2021-08-28)

- Added the *compare command*.
- Added the *manage_diffs_cmd* and *report_diffs_cmd* settings.
- Added the *run_before_first_backup* and *run_after_last_backup* settings.
- Allow files listed by *manifest* command to be constrained to those contained within a path.
- Allow relative dates to be specified on the *extract*, *manifest*, *mount* and *restore* commands.
- Allow *BORG_PASSPHRASE*, *BORG_PASSPHRASE_FD*, or *BORG_PASSCOMMAND* to dominate over *Emborg* passphrase settings.

10.8.16 1.24 (2021-07-05)

- Added *healthchecks_url* and *cronhub_url* settings.

10.8.17 1.23 (2021-07-01)

- Fix missing dependency.

10.8.18 1.22 (2021-06-21)

- Added support for healthchecks.io monitoring service.
- Added support for cronhub.io monitoring service.

10.8.19 1.21 (2021-03-11)

- Made extensive changes to *manifest* command to make it more flexible
  - colorized the output based on file health (green implies healthy, red implies unhealthy)
  - added `--no-color` option to *manifest* to suppress colorization
  - added *colorscheme* setting.
  - added *manifest_default_format* setting.
  - added support for *Borg list* command field names for both reporting and sorting.
  - added *Emborg* variants to some of the *Borg* field names.
  - added `--show-formats` command line option.
  - added `--format` command line option.
  - added `--sort-by-field` command line option.
  - change predefined formats to use fields that render faster

  Warning: These changes are not backward compatible. If you have a *manifest_formats* setting from a previous version, it may need to be updated.

- It is now an error for *prefix* setting to contain `{{now}}`.
- *Settings* command will now print a single setting value if its name is given.

10.8.20 1.20 (2021-02-13)

- Add `--progress` command-line option and *show_progress* option to the *create* command.

10.8.21 1.19 (2021-01-02)

- Added `--list` command-line option to the *prune* command.

10.8.22 1.18 (2020-07-19)

- Added `--repo` option to *delete* command.
- Added `--relocated` global command-line option.
- *Emborg* now automatically confirms to *Borg* that you know what you are doing when you delete a repository or repair an archive.

10.8.23 1.17 (2020-04-15)

- *Borg* command allows archive to be added to `@repo`.
- Added *encoding* setting.

10.8.24 1.16 (2020-03-17)

- Refinements and bug fixes.

10.8.25 1.15 (2020-03-06)

- Improve messaging from *emborg-overdue*
- *Configs* command now outputs default configuration too.
- Some commands now use first subconfig when run with a composite configuration rather than terminating with an error.
- Added *show_stats* setting.
- Added `--stats` option to *create*, *delete* and *prune* commands.
- Added `--list` option to *create*, *extract* and *restore* commands.
- Added sorting and formatting options to *manifest* command.
- Added *manifest_formats* setting.
- Renamed `--trial-run` option to `--dry-run` to be more consistent with *Borg*.
- Add *files* and *f* aliases to *manifest* command.
- Added *working_dir* setting.
- Added *do_not_expand* setting.
- Added *exclude_nodump* setting
- Added *patterns* and *patterns_from* settings.
- *Emborg* lock file is now ignored if the process it references is no longer running
- Support `--repair` option on *check command*.

10.8.26 1.14 (2019-12-31)

- Remove debug message accidentally left in *emborg-overdue*

10.8.27 1.13 (2019-12-31)

- Enhance *emborg-overdue* to work on clients as well as servers

10.8.28 1.12 (2019-12-25)

- Added *default_mount_point* setting.
- Fixed some issues with *borg* command.
- Added `--oldest` option to *due* command.

10.8.29 1.11 (2019-11-27)

- Bug fix release.

10.8.30 1.10 (2019-11-11)

- Bug fix release.

10.8.31 1.9 (2019-11-08)

- Added ability to check individual archives to the *check* command.
- Made latest archive the default for *check* command.
- Allow *exclude_from* setting to be a list of file names.

10.8.32 1.8 (2019-10-12)

- Remove duplicated commands.

10.8.33 1.7 (2019-10-07)

- Fixed bug that involved the Boolean Borg settings (*one_file_system*, *exclude_caches*, ...)

10.8.34 1.6 (2019-10-04)

- Added *restore* command.
- Added *verbose* setting.

10.8.35 1.5 (2019-09-30)

- Added composite configurations.
- Added support for multiple backup configurations in a single repository.
- Added *prefix* and *exclude_from* settings.
- Provide default value for *archive* setting.
- Add `--all` command line option to *mount* command.
- Add `--include-external` command line option to *check*, *list*, *mount*, and *prune* commands.
- Add `--sort` command line option to *manifest* command.
- Add `--latest` command line option to *delete* command.
- Added `--quiet` command line option
- *umount* command now deletes directory used as mount point.
- Moved log files to ~/.local/share/emborg (run 'mv ~/.config/emborg/*.{log,lastbackup}* ~/.local/share/emborg' before using this version).

10.8.36 1.4 (2019-04-24)

- Added *ssh_command* setting
- Added `--fast` option to *info* command
- Added *emborg-overdue* executable
- Allow *run_before_backup* and *run_after_backup* to be simple strings

10.8.37 1.3 (2019-01-16)

- Added the raw *borg* command.

10.8.38 1.2 (2019-01-16)

- Added the *borg_executable* and *passcommand* settings.

10.8.39 1.1 (2019-01-13)

- Improved and documented API.
- Creates the settings directory if it is missing and add example files.
- Added `--mute` command line option.
- Support multiple email addresses in *notify*.
- Added warning if settings file is world readable and contains a passphrase.

10.8.40 1.0 (2019-01-09)

- Added *remote_path* setting.
- Formal public release.

10.8.41 0.3 (2018-12-25)

- Initial public release (beta).

10.8.42 0.0 (2018-12-05)

- Initial release (alpha).
- genindex