Backups with Obnam

Lars Wirzenius (liw@liw.fi)

Version obnam-1.21-37-g0d2af29

Contents

Chapter 1

Introduction

> ... backups? did someone talk about backups? I'm sure I heard someone mention backups here somewhere. Backups! BACKUPS! BACKUPS ARE AWESOME!

That's a direct quote from my IRC history. I find backups quite interesting, particularly from an implementation point of view, and I may sometimes obsess about them a little bit. This is why I've written my own backup software. It's called Obnam. This is its manual.

I'm unusual: most people find backups boring at best, and tedious most of the time. When I talk with people about backups, the usual reaction is "um, I know I should". There are a lot of reasons for this. One is that backups are a lot like insurance: you have to spend time, effort, and money up front to have any use for them. Another is that the whole topic is scary: you have to think about when things go wrong, and that puts people off. A third reason is that while there are lots of backup tools and methods, it's not always easy to choose between them.

This manual is for the Obnam program, but it tries to be useful to everyone thinking about backups.

Chapter 2

TL;DR: README FIRST: A quick tour of Obnam

You probably only need to read this chapter.

This chapter gives a quick introduction to the most important parts of Obnam. The rest of the book is basically a verbose version of this chapter. You should start by reading this chapter, then pretend you've read the rest, and everyone will look at you in awe at cocktail parties. I promise, nobody else will have read the rest of the book either, so there's no risk of getting caught.

2.1 Configuration

Obnam does not require a configuration file, and you can configure everything using command line options. You can, however, use a configuration file: save it as `~/.obnam.conf` and make it have content like this:

```
[config]
repository = sftp://your.server/home/youruser/backups/
log = /home/liw/obnam.log
```

The examples below assume you have created a configuration file, so that options do not need to be repeated every time.

You probably want to enable the `log` setting, so that if there is a problem, you can find out all the information available to fix it from the log file.

2.2 Initial backup

Your first backup will be pretty big, and will take a long time. A long backup may crash, but that is not a problem: Obnam makes a **checkpoint** every one hundred megabytes or so.

```
obnam backup $HOME
```

2.3 Incremental backups

When you've made your initial, full backup (possibly in stages), you can back up any changes simply by running Obnam again:

```
obnam backup $HOME
```

This will back up all new files, and any changed files. It will also record which files have been deleted since the previous backup.

You can run Obnam as often as you like. Only the changes from the previous run are backed up.

2.4 Multiple clients in one repository

You can backup multiple clients to a single repository by providing the option –client-name= when running the program. Backup sets will be kept separate, but data de-duplication will happen across all the sets.

2.5 Removing old generations

Eventually your backup repository will grow so big you'll want to remove some old generations. The Obnam operation is called forget:

```
obnam forget --keep=30d
```

This would keep one backup from each of the last thirty calendar days, counting from the newest backup (not current time). If you've backed up several times during a day, only the latest generation from that day is kept.

Any data that is part of a generation that is to be kept will remain in the repository. Any data that exists only in those generations that is to be forgotten gets removed.

2.6 Restoring data

You will hopefully never need this, but the whole point of having backups is to restore data in case of a disaster.

```
obnam restore --to=/var/tmp/my-recovery $HOME
```

The above command will restore your entire home directory to `/var/tmp/my-recovery`, from the latest backup generation. If you only need some particular directory or file, you can specify that instead:

```
obnam restore --to=/var/tmp/my-recover $HOME/Archive/receipts
```

If you can't remember the name of the file you need, use `obnam ls`:

```
obnam ls > /var/tmp/my-recovery.list
```

This will output the contents of the backup generation, in a format similar to `ls -lAR`. Save it into a file and browse that. (It's a fairly slow command, so it's comfortable to save to a file.)

2.7 Using encryption

Obnam can use the GnuPG program to encrypt the backup. To enable this, you need to have or create a PGP key, and then configure Obnam to use it:

```
[config]
encrypt-with = CAFEBABE
```

Here, `CAFEBABE` is the **key identifier** for your key, as reported by GnuPG. You need to have `gpg-agent` or equivalent software configured, for now, because Obnam has no way to ask for or configure the passphrase.

After this, Obnam will automatically encrypt and decrypt data.

Note that if you encrypt your backups, you'll want to back up your GPG key in some other way. You can't restore any files from the obnam backup without it, so you can't rely on the same obnam backup to back up the GPG key itself. Back up your passphrase-encrypted GPG key somewhere else, and make sure you have a passphrase strong enough to stand up to offline brute-force attacks. Remember that if you lose access to your GPG key, your entire backup becomes useless.

If you enable encryption after making backups, you need to start over with a new repository. You can't mix encrypted and unencrypted backups in the same repository.

(There are a bunch of Obnam commands for administering encryption. You won't need them, unless you share the same repository with several machines. In that case, you should read the manual page.)

Chapter 3

You know you should

This chapter is philosophical and theoretical about backups. It discusses why you should back up, various concepts around backups, what kinds of things you should think about when setting up backups and what to do in the long term (verification, etc). It also discusses some assumptions Obnam makes and some constraints it imposes.

3.1 Why backup?

FIXME: Add some horror stories here about why backups are important. With references/links.

3.2 Backup concepts

This section covers core concepts in backups, and defines some terminology used in this book.

Live data is the data you work with or keep. It's the files on your hard drive: the documents you write, the photos you save, the unfinished novels you wish you'd finish.

Most live data is **precious** in that you'll be upset if you lose it. Some live data is not precious: your web browser cache probably isn't, for example. This distinction can let you limit the amount of data you need to back up, which can significantly reduce your backup costs.

A **backup** is a spare copy of your live data. If you lose some or all of your live data, you can get it back ("**restore**") from your backup. The backup copy is,

by practical necessity, older than your live data, but if you made the backup recently enough, you won't lose much.

Sometimes it's useful to have more than one old backup copy of your live data. You can have a sequence of backups, made at different times, giving you a **backup history**. Each copy of your live data in your backup history is a **generation**. This lets you retrieve a file you deleted a long time ago, but didn't realise you needed until now. If you only keep one backup version, you can't get it back, but if you keep, say, a daily backup for a month, you have a month to realise you need it, before it's lost forever.

The place your backups are stored is the **backup repository**. You can use many kinds of **backup media** for backup storage: hard drives, tapes, optical disks (DVD-R, DVD-RW, etc), USB flash drives, online storage, etc. Each type of medium has different characteristics: size, speed, convenience, reliability, price, which you'll need to balance for a backup solution that's reasonable for you.

You may need multiple backup repositories or media, with one of them located **off-site**, away from where your computers normally live. Otherwise, if you house burns down, you'll lose all your backups too.

You need to **verify** that your backups work. It would be awkward to go to the effort and expense of making backups and then not be able to restore your data when you need to. You may even want to test your **disaster recovery** by pretending that all your computer stuff is gone, except for the backup media. Can you still recover? You'll want to do this periodically, to make sure your backup system keeps working.

There is a very large variety of **backup tools**. They can be very simple and manual: you can copy files to a USB drive using your file manager, once a blue moon. They can also be very complex: enterprise backup products that cost huge amounts of money and come with a multi-day training package for your sysadmin team, and which require that team to function properly.

You'll need to define a **backup strategy** to tie everything together: what live data to back up, to what medium, using what tools, what kind of backup history to keep, and how to verify that they work.

3.3 Backup strategies

You've set up a backup repository, and you have been backing up to it every day for a month now: your backup history is getting long enough to be useful. Can you be happy now?

Welcome to the world of threat modelling. Backups are about insurance, of mitigating small and large disasters, but disasters can strike backups as well. When are you so safe that no disaster will harm you?

There is always a bigger disaster waiting to happen. If you backup to a USB drive on your work desk, and someone breaks in and steals both your computer and the USB drive, the backups did you no good.

You fix that by having two USB drives, and you keep one with your computer and the other in a bank vault. That's pretty safe, unless there's an earth quake that destroys both your home and the bank.

You fix that by renting online storage space from another country. That's quite good, except there's a bug in the operating system that you use, which happens to be the same operating system the storage provider uses, and hackers happen to break into both your and their systems, wiping all files.

You fix that by hiring a 3D printer that prints slabs of concrete on which your data is encoded using QR codes. You're safe until there's a meteorite hits Earth and destroys the entire civilisation.

You fix that by sending out satellites with copies of your data, into stable orbits around all nine planets (Pluto is too a planet!) in the solar system. Your data is safe, even though you yourself are dead from the meteorite, until the Sun goes supernova and destroys everything in the system.

There is always a bigger disaster. You have to decide which ones are likely enough that you want to consider them, and also decide what the acceptable costs are for protecting against them.

A short list of scenarios for thinking about threats:

- What if you lose your computer?
- What if you lose your home and all of its contents?
- What if the area in which you live is destroyed?
- What if you have to flee your country?

These questions do not cover everything, but they're a start. For each one, think about:

- Can you live with your loss of data? If you don't restore your data, does it cause a loss of memories, or some inconvenience in your daily life, or will it make it nearly impossible to go back to living and working normally? What data do you care most about?
- How much is it worth to you to get your data back, and how fast do you want that to happen? How much are you willing to invest money and effort to do the initial backup, and to continue backing up over time? And for restores, how much are you willing to pay for that? Is it better for you to spend less on backups, even if that makes restores slower, more expensive, and more effort? Or is the inverse true?

The threat modelling here is about safety against accidents and natural disasters. Threat modelling against attacks and enemies is similar, but also different, and will be the topic of the next episode in the adventures of Bac-Kup.

3.4 Backups and security

You're not the only one who cares about your data. A variety of governments, corporations, criminals, and overly curious snoopers are probably also interested. (It's sometimes hard to tell them apart.) They might be interested to find evidence against you, blackmail you, or just curious about what you're talking about with your other friends.

They might be interested in your data from a statistical point of view, and don't particularly care about you specifically. Or they might be interested only in you.

Instead of reading your files and e-mail, or looking at your photos and videos, they might be interested in preventing your access to them, or to destroy your data. They might even want to corrupt your data, perhaps by planting child porn in your photo archive.

You protect your computer as well as you can to prevent these and other bad things from happening. You need to protect your backups with equal care.

If you back up to a USB drive, you should probably make the drive be encrypted. Likewise, if you back up to online storage. There are many forms of encryption, and I'm unqualified to give advice on this, but any of the common, modern ones should suffice except for quite determined attackers.

Instead of, or in addition to, encryption, you could ensure the physical security of your backup storage. Keep the USB drive in a safe, perhaps, or a safe deposit box.

The multiple backups you need to protect yourself against earthquakes, floods, and roving gangs of tricycle-riding clowns, are also useful against attackers. They might corrupt your live data, and the backups at your home, but probably won't be able to touch the USB drive encased in concrete and buried in the ground at a secret place only you know about.

The other side of the coin is that you might want to, or need to, ensure others do have access to your backed up data. For example, if the clown gang kidnaps you, your spouse might need access to your backups to be able to contact your MI6 handler to ask them to rescue you. Arranging safe access to (some) backups is an interesting problem to which there are various solutions. You could give your spouse the encryption passphrase, or give the passphrase to a trusted friend or your lawyer. You could also use something like libgfshare to escrow encryption keys more safely.

3.5 Backup storage media considerations

This section discusses possibilities for backup storage media, and their various characteristics, and how to choose the suitable one for oneself.

There are a lot of different possible storage media. Perhaps the most important ones are:

- Magnetic tapes of various kinds.
- Hard drives: internal vs external, spinning magnetic surfaces vs SSDs vs memory sticks.
- Optical disks: CD, DVD, Blu-ray.
- Online storage of various kinds.
- Paper.

We'll skip more exotic or unusual forms, such as microfilm.

Magnetic tapes are traditionally probably the most common form of backup storage. They can be cheap per gigabyte, but tend to require a fairly hefty initial investment in the tape drive. Much backup terminology comes from tape drives: full backup vs incremental backup, especially. Obnam doesn't support tape drives at all.

Hard drives are a common modern alternative to tapes, especially for those who do not wish pay for a tape drive. Hard drives have the benefit of every bit of backup being accessible at the same speed as any other bit, making finding a particular old file easier and faster. This also enables **snapshot backups**, which is the model Obnam uses.

Different types of hard drives have different characteristics for reliability, speed, and price, and they may fluctuate fairly quickly from week to week and year to year. We won't go into detailed comparisons of all the options. From Obnam's point of view, anything that can look like a hard drive (spinning rust, SSD, USB flash memory stick, or online storage) is usable for storing backups, as long as it is re-writable.

Optical disks, particularly the kind that are write-once and can't be updated, can be used for backup storage, but they tend to be best for full backups that are stored for long periods of time, perhaps archived permanently, rather than for a actively used backup repository. Alternatively, they can be used as a kind of tape backup, where each tape is only ever used once. Obnam does not support optical drives as backup storage.

Paper likewise works better for archival purposes, and only for fairly small amounts of data. However, a backup printed on good paper with archival ink can last decades, even centuries, and is a good option for small, but very precious data. As an example, personal financial records, secret encryption keys, and love letters from your spouse. These can be printed either normally (preferably in a

font that is easy to OCR), or using two-dimensional barcode (e.g, QR). Obnam doesn't support these, either.

Obnam only works with hard drives, and anything that can simulate a read/writable hard drive, such as online storage. By amazing co-incidence, this seems to be sufficient for most people.

3.6 Glossary

- **backup**: a separate, safe copy of your live data that will remain intact even if the primary copy gets destroyed, deleted, or wrongly modified
- **corruption**: unwanted modification to (backup) data
- **disaster recovery**: what you do when something goes wrong
- **full backup**: a fresh backup of all precious live data
- **generation**: a backup in a series of backups of the same live data, to give historical insight
- **history**: all the backup generations
- **incremental backup**: a backup of any changes (new files, modified files, deletions) compared to a previous backup generation (either the previous full backup, or the previous incremental backup); usually, you can't remove a full backup without removing all of the incremental backups that depend on it
- **live data**: all the data you have
- **local backup**: a backup repository stored physically close to the live data
- **media**, **backup media**, **storage media**: where a backup repository is stored
- **off-site backup**: a backup repository stored physically far away from the live data
- **precious data**: all the data you care about; cf. live data
- **repository**: the location where backup data is stored
- **restore**: retrieving data from a backup repository
- **root**, **backup root**: a directory that is to be backed up, including all files in it, and all its subdirectories
- **snapshot backup**: an alternative to full/incremental backups, where every backup generation is effectively a full backup of all the precious live data, and can be restored and removed as easily as any other generation
- **strategy**, **backup strategy**: a plan for how to make sure your data is safe even if the dinosaurs return in space ships to re-take world now that the ice age is over
- **verification**: making sure a backup system works and that data actually can be restored from backups and that the backups have not become corrupted

Chapter 4

Installing Obnam

This chapter explains how to install Obnam. It is not a very extensive set of instructions, yet. In particular, it really only caters to Debian users. Instructions for other systems would be very much welcome.

4.1 Debian

It is easiest to install Obnam on a Debian system. If you're running Debian `wheezy` or a later release, Obnam is included and you can just install it:

```
apt-get install obnam
```

There may be a newer version of Obnam on the author's site. The rest of this section explains how to install from there.

Add the following line to your `/etc/apt/sources.list` file:

```
deb http://code.liw.fi/debian squeeze main
```

Then run the following commands as root:

- `apt-get update`
- `apt-get install obnam`

The commands will complain that the PGP key used to sign the archive is not known to apt. You can either ignore this, or add the key from http://code.liw.fi/apt.asc to your key, after suitable verification.

4.2 Other systems

For other systems, you need to install from sources. See the `README` file in the source tree for instructions.

Chapter 5

Backing up

This chapter discusses the various aspects of making backups with Obnam.

5.1 Your first backup

Let's make a backup! To walk through the examples in this directory, you need to have some live data to backup. The examples use specific filenames for this. You'll need to adapt the examples to your own files. The examples assume your home directory is `/home/tomjon`, and that you have a directory called `Documents` in your home directory for your documents. Further, it assumes you have a USB drive mounted at `/media/backups`, and that you will be using a directory `tomjon-repo` on that drive as the backup repository.

With those assumptions, here's how you would backup your documents:

```
obnam backup -r /media/backups/tomjon-repo ~/Documents
```

That's all. It will take a little while, if you have a lot of documents, but eventually it'll look something like this:

```
Backed up 11 files (of 11 found),
uploaded 97.7 KiB in 0s at 647.2 KiB/s average speed
```

(In reality, the above text will be all on one line, but that didn't fit in this manual's line width.)

This tells you that Obnam found a total of eleven files, of which it backed up all eleven. The files contained a total of about a hundred kilobytes of data, and that the upload speed for that data was over six hundred kilobytes per second.

The actual units are using IEC prefixes, which are base-2, to avoid ambiguity. See Wikipedia on kibibytes for more information.

Your first backup run should probably be quite small to see that all settings are right without having to wait a long time. You may want to choose a small directory to start with, instead of your entire home directory.

5.2 Your second backup

Once you've run your first backup, you'll want to run a second one. It's done the same way:

```
obnam backup -r /media/backups/tomjon-repo ~/Documents
```

Note that you don't need to tell Obnam whether you want a full backup or an incremental backup. Obnam makes each backup generation be a snapshot of the data at the time of the backup, and doesn't make a difference between full and incremental backups. Each backup generation is equal to each other backup generation. This doesn't mean that each generation will store all the data separately. Obnam makes sure each new generation only backs up data that isn't already in the repository. Obnam finds that data in any file in any previous generation, amongst all the clients sharing the same repository.

We'll later cover how to remove backup generations, and you'll learn that Obnam can remove any generation, even if it shares some of the data with other generations, without those other generations losing any data.

After you've your second backup generation, you'll want to see the generations you have:

```
$ obnam generations -r /media/backups/tomjon-repo
2   2014-02-05 23:13:50 .. 2014-02-05 23:13:50 (14 files, 100000 bytes)
5   2014-02-05 23:42:08 .. 2014-02-05 23:42:08 (14 files, 100000 bytes)
```

This lists two generations, which have the identifiers 2 and 5. Note that generation identifiers are not necessarily a simple sequence like 1, 2, 3. This is due to how some of the internal data structures of Obnam are implemented, and not because its author in any way thinks it's fun to confuse people.

The two time stamps for each generation are when the backup run started and when it ended. In addition, for each generation is a count of files in that generation (total, not just new or changed files), and the total number of bytes of file content data they have.

5.3 Choosing what to backup, and what not to backup

Obnam needs to be told what to back up, by giving it a list of directories, known as backup roots. In the examples in this chapter so far, we've used the directory `~/Documents` (that is, the directory `Documents` in your home directory) as the backup root. There can be multiple backup roots:

```
obnam -r /media/backups/tomjon-repo ~/Documents ~/Photos
```

Everything in the backup root directories gets backed up – unless it's explicitly excluded. There are several ways to exclude things from backups:

- The `--exclude` setting uses regular expressions that match the full pathname of each file or directory: if the pathname matches, the file or directory is not backed up. In fact, Obnam pretends it doesn't exist. If a directory matches, then any files and sub-directories also get excluded. This can be used, for example, to exclude all MP3 files (`--exclude='\.mp3$'`).
- The `--exclude-caches` setting excludes directories that contain a special "cache tag" file called `CACHEDIR.TAG`, that starts with a specific sequence of bytes. Such a tag file can be created in, for example, a Firefox or other web browser cache directory. Those files are usually not important to back up, and tagging the directory can be easier than constructing a regular expression for `--exclude`.
- The `--one-file-system` setting excludes any mount points and the contents of the mounted filesystem. This is useful for skipping, for example, virtual filesystems such as `/proc`, remote filesystems mounted over NFS, and Obnam repositories mounted with `obnam mount` (which we'll cover in the next chapter).

In general it is better to back up too much rather than too little. You should also make sure you know what is and isn't backed up. The `--pretend` option tells Obnam to run a backup, except it doesn't change anything in the backup repository, so it's quite fast. This way you can see what would be backed up, and tweak exclusions as needed.

5.4 Storing backups remotely

You probably want to store at least one backup remotely, or "off site". Obnam can make backups over a network, using the SFTP protocol (part of SSH). You need the following to achieve this:

- A **server** that you can access over SFTP. This can be a server you own, a virtual machine ("VPS") you rent, or some other arrangement. You could, for example, have a machine at a friends' place, in exchange for having one of their machines at your place, so that you both can backup remotely.
- An **ssh key** for logging into the server. Obnam does not currently support logging in via passwords.
- Enough disk space on the server to hold your backups.

Obnam only uses the SFTP part of the SSH connection to access the server. To test whether it will work, you can run this command:

```
sftp USER@SERVER
```

Change `USER@SERVER` to be your actual user and address for your server. This should say something like `Connected to localhost.` and you should be able to run the `ls -la` command to see a directory list of files on the remote side.

Once all of that is set up correctly, to get Obnam to use the server as a backup repository, you only need to give an SFTP URL:

```
obnam -r sftp://USER@SERVER/~/my-precious-backups
```

For details on SFTP URLs, see the next section.

5.5 URL syntax

Whenever obnam accepts a URL, it can be either a local pathname, or an SFTP URL. An SFTP URL has the following form:

```
sftp://[user@]domain[:port]/path
```

where `domain` is a normal Internet domain name for a server, `user` is your username on that server, `port` is an optional numeric port number, and `path` is a pathname on the server side. Like **bzr**(1), but unlike the SFTP URL standard, the pathname is absolute, unless it starts with `/~/` in which case it is relative to the user's home directory on the server.

Examples:

- `sftp://liw@backups.pieni.net/~/backup-repo` is the directory `backup-repo` in the home directory of the user `liw` on the server `backups.pieni.net`. Note that we don't need to know the absolute path of the home directory.

- `sftp://root@my.server.example.com/home` is the directory `/home` (note absolute path) on the server `my.server.example.com`, and the `root` user is used to access the server.
- `sftp://foo.example.com:12765/anti-clown-society` is the directory `/anti-clown-society` on the server `foo.example.com`, accessed via the port 12765.

You can use SFTP URLs for the repository, or the live data (`--root`), but note that due to limitations in the protocol, and its implementation in the paramiko library, some things will not work very well for accessing live data over SFTP. Most importantly, the handling of of hardlinks is rather suboptimal. For live data access, you should not end the URL with /~/ and should append a dot at the end in this special case.

5.6 Pull backups

Obnam can also access the live data over SFTP, instead of via the local filesystem. This means you can run Obnam on, say, your desktop machine to backup your server, or on your laptop to backup your phone (assuming you can get an SSH server installed on your phone). Sometimes it is not possible to install Obnam on the machine where the live data resides, and then it is useful to do a **pull backup** instead: you run Obnam on a different machine, and read the live data over the SFTP protocol.

To do this, you specify the live data location (the `root` setting, or as a command line argument to `obnam backup`) using an SFTP URL. You should also set the client name explicitly. Otherwise Obnam will use the hostname of the machine on which it runs as the name, and this can be highly confusing: the client name is `my-laptop` and the server is `down-with-clowns` and Obnam will store the backups as if the data belongs to `my-laptop`.

It gets worse if you backup your laptop as well to the same backup repository. Then Obnam will store both the server and the laptop backups using the same client name, resulting in much confusion to everyone.

Example:

```
obnam backup -r /mnt/backups sftp://server.example.com/home \
    --client-name=server.example.com
```

5.7 Configuration files: a quick intro

By this time you may have noticed that Obnam has a number of configurable settings you can tweak in a number of ways. Doing it on the command line is

always possible, but then you get quite long command lines. You can also put them into a configuration file.

Every command line option Obnam knows can be set in a configuration file. Later in this manual there is a whole chapter that covers all the details of configuration files, and all the various settings you can use. For now, we'll give a quick introduction.

An Obnam configuration looks like this:

```
[config]
repository = /media/backup/tomjon-repo
root = /home/liw/Documents, /home/liw/Photos
exclude = \.mp3$
exclude-caches = yes
one-file-system = no
```

This form of configuration file is commonly known as an "INI file", from Microsoft Windows `.INI` files. All the Obnam settings go into a section titles `[config]`, and each setting has the same name as the command line option, but without the double dash prefix. Thus, it's `--exclude` on the command line and `exclude` in the configuration file.

Some settings can have multiple values, such as `exclude` and `root`. The values are comma separated. If there's a lot of values, you can split them on multiple lines, where the second and later lines are indented by space or TAB characters.

That should get you started, and you can reference the "Obnam configuration files and settings" chapter for all the details.

5.8 When your precious data is very large

When your precious data is very large, the first backup may a very long time. Ditto, if you get a lot of new precious data for a later backup. In these cases, you may need to be very patient, and just let the backup take its time, or you may choose to start small and add to the backups a bit at a time.

The patient option is easy: you tell Obnam to backup everything, set it running, and wait until it's done, even if it takes hours or days. If the backup terminates prematurely, e.g., because of a network link going down, you won't have to start from scratch thanks to Obnam's checkpoint support. Every gigabyte or so (by default) Obnam stops a backup run to create a checkpoint generation. If the backup later crashes, you can just re-run Obnam and it will pick up from the latest checkpoint. This is all fully automatic, you don't need to do anything for it to happen. See the `--checkpoint` setting for choosing how often the checkpoints should happen.

Note that if Obnam doesn't get to finish, and you have to re-start it, the scanning starts over from the beginning. The checkpoint generation does not contain enough state for Obnam to continue scanning from the latest file in the checkpoint: it'd be very complicated state, and easily invalidated by filesystem changes. Instead, Obnam scans all files, but most files will hopefully not have changed since the checkpoint was made, so the scanning should be relatively fast.

The only problem with the patient option is that your most precious data doesn't get backed up while all your large, but less precious data is being backed up. For example, you may have a large amount of downloaded videos of conference presentations, which are nice, but not hugely important. While those get backed up, your own documents do not get backed up.

You can work around this by initially excluding everything except the most precious data. When that is backed up, you gradually reduce the excludes, re-running the backup, until you've backed up everything. As an example, your first backup might have the following configuration:

```
obnam backup -r /media/backups/tomjon-repo ~ \
    --exclude ~/Downloads
```

This would exclude all downloaded files. The next backup run might exclude only video files:

```
obnam backup -r /media/backups/tomjon-repo ~ \
    --exclude ~/Downloads/'.*\.mp4$'
```

After this, you might reduce excludes to allow a few videos, such as those whose name starts with a specific letter:

```
obnam backup -r /media/backups/tomjon-repo ~ \
    --exclude ~/Downloads/'[^b-zB-Z].*\.mp4$'
```

Continue allowing more and more videos until they've all been backed up.

5.9 De-duplication

Obnam de-duplicates the data it backs up, across all files in all generations for all clients sharing the repository. It does this by breaking up all file data into bits called chunks. Every time Obnam reads a file and gets a chunk together, it looks into the backup repository to see if an identical chunk already exists. If it does, Obnam doesn't need to upload the chunk, saving space, bandwidth, and time.

De-duplication in Obnam is useful in several situations:

- When you have two identical files, obviously. They might have different names, and be in different directories, but contain the same data.
- When a file keeps growing, but all the new data is added at the end. This is typical for log files, for example. If the leading chunks are unmodified, only the new data needs to be backed up.
- When a file or directory is renamed or moved. If you decide that the English name for the `Photos` directory is annoying and you want to use the Finnish `Valokuvat` instead, you can rename that in an instant. However, without de-duplication, you then have to backup all your photos again.
- When all a team works on the same things, and everyone has copies of the same files, the backup repository only needs one copy of each file, rather than one per team member.

De-duplication in Obnam isn't perfect. The granularity of finding duplicate data is quite coarse (see the `--chunk-size` setting), and so Obnam often doesn't find duplication when it exists, when the changes are small.

De-duplication isn't useful in the following scenarios:

- A file changes such that things move around within the file. The (current) Obnam de-duplication is based on non-overlapping chunks from the beginning of a file. If some data is inserted, Obnam won't notice that the chunks have shifted around. This can happen, for example, for disk or ISO images.

- Files with duplicate data that is not on a chunk boundary. For example, emails with large attachments. Each email recipient gets different `Received` headers, which shifts the body and attachments by different amounts. As a result, Obnam won't notice the duplication.

- Data in compressed files, such as `.zip` or `.tar.xz` files. Obnam doesn't know about the file compression, and only sees the compressed version of the data. Thus, Obnam won'd de-duplicate it.

A future version of Obnam will hopefully improve the de-duplication algorithms. If you see this optimistic paragraph in a version of Obnam released in 2017 or later, please notify the maintainers. Thank you.

5.10 De-duplication and safety against checksum collisions

This is a bit of a scary topic, but it would be dishonest to not discuss it at all. Feel free to come back to this section later.

Obnam uses the MD5 checksum algorithm for recognising duplicate data chunks. MD5 has a reputation for being unsafe: people have constructed files that are different, but result in the same MD5 checksum. This is true. MD5 is not considered safe for security critical applications.

Every checksum algorithm can have collisions. Changing Obnam to use, say, SHA1, SHA2, or the as new SHA3 algorithm would not remove the chance of collisions. It would reduce the chance of accidental collisions, but the chance of those is already so small with MD5 that it can be disregarded. Or put in another way, if you care about the chance of accidental MD5 collisions, you should be caring about accidental SHA1, SHA2, or SHA3 collisions as well.

Apart from accidental collisions, there are two cases where you should worry about checksum collisions (regardless of algorithm).

First, if you have an enemy who wishes to corrupt your backed up data, they may replace some of the backed up data with other data that has the same checksum. This way, when you restore, your data is corrupted without Obnam noticing.

Second, if you're into researching checksum collisions, you're likely to have files that cause checksum collisions, and in that case, if you restore after a catastrophe, you probably want to get the files back intact, rather having Obnam confuse one with the other.

To deal with these situations, Obnam has three de-duplication modes, set using the `--deduplicate` setting:

- The default mode, `fatalist`, assumes checksum collisions do not happen. This is a reasonable compromise between performance, safety, and security for most people.
- The `verify` mode assumes checksum collisions do happen, and verifies that the already backed up chunk is identical to the chunk to be backed up, by comparing the actual data. Doing this requires downloading the chunk from the backup repository, which can be quite slow, since checksums will often match. This is a useful mode if you have very fast access to the backup repository, and want to de-duplicate, such as when the backup repository is on a locally connected hard drive.
- The `never` mode turns off de-duplication completely. This is useful if you're worried about checksum collisions, and do not require de-duplication.

There is, unfortunately, no way to get both de-duplication that is invulnerable to checksum collision and is fast even when accessing the backup repository is slow. The only way to be invulnerable is to compare the data, and if downloading the data from the repository is slow, then the comparison will take significant time.

5.11 Locking

Multiple clients can share a repository, and to prevent them from trampling on each other, they lock parts of the repository while working. The "Sharing a repository between multiple clients" chapter will discuss this in more detail.

If Obnam terminates abruptly, even if there's only one client ever using the repository, the lock may stay around and prevent that one client for making new backups. The termination may be due to the network connection breaking, or due to a bug in Obnam. It can also happen if Obnam is interrupted by the user before it's finished.

The Obnam command `force-lock` deals with this situation. It is dangerous, though. If you force open a lock that is in active use by any running Obnam instance, on any client machine using that repository, there will likely to be some stepping of toes. The result may, in extreme cases, even result in repository corruption. So be careful.

If you've decided you can safely do it, this is an example of how to do it:

```
obnam -r /media/backups/tomjon-repo force-lock
```

It is not currently possibly to only break locks related to one client.

5.12 Consistency of live data

Making a backup can take a good while. While the backup is running, the filesystem may change. This leads to the snapshot of data Obnam presents as a backup generation being internally inconsistent. For example, before a backup you might have two files, A and B, which need to be kept in sync. You run a backup, and while it's happening, you change A, and then B. However, you're unlucky, and Obnam manages to backup A before you save your changes, and B after you save changes to that. The backup generation now has versions of A and B that are not synchronised. This is bad.

This can be dealt with in various ways, depending on the circumstances. Here's a few:

- The Logical Volume Manager (LVM) provides snapshots. You can set up your backups so that they first create a snapshot of each logical volume to be backed up, run the backup, and delete the snapshot afterwards. This prevents anyone from modifying the files in the snapshot, but allows normal use to continue while the backup happens.
- A similar thing can be done using the btrfs filesystem and its subvolumes.

- You can shut down the system, reboot it into single user mode, and run the backup, before rebooting back into normal mode. This is not a good way to do it, but it is the safest way to get a consistent snapshot of the filesystem.

Note that filesystem level snapshots can't really guarantee a consistent view of the live data. An application may be in the middle of writing a file, or set of files, when the snapshot is being made. To some extent this indicates an application bug, but knowing that doesn't let you sleep better.

Usually, though, most systems have enough idle time that a consistent backup snapshot can happen during that time. For a laptop, for example, a backup can be run while the user is elsewhere, instead of actively using the machine.

Part of your backup verification suite should check that the data in a backup generation is internally consistent, if that can be done. Otherwise, you'll either have to analyse the applications you use, or trust they're not too buggy.

If you didn't understand this section, don't worry and be happy and sleep well.

Chapter 6

Restoring from backups

The worst has happened! Your cat got confused between its litter box and your hard drive! Your goat deleted your most important document ever! Woe be you!

Let's stay calm. This is why you have backups. There's no need for exclamation marks. Take a deep breath, have a cup of tea, and all will be well.

There's two different approaches for restoring data with Obnam. One relies on a FUSE filesystem, which is a very nice piece of technology that allows Obnam to let you view your backups as just another directory. It is the preferred way, but it is not always available, so Obnam also provides a more primitive, less easy to use method.

6.1 Oh no! It's all FUSEd together

The `obnam mount` command lets you look at your backups as if they were just another directory. This requires that you have FUSE setup. See the installation chapter for details on that. Most modern Linux desktops have this out of the box.

```
mkdir ~/backups
obnam mount --to ~/backups
```

Run the above command, and then look at the `~/backups` directory. You'll see something like this:

```
$ ls -l ~/backups
total 12
drwxr-xr-x 24 root root 4096 Feb 11 21:41 2
```

```
drwxr-xr-x 24 root root 4096 Feb 11 21:41 5
lrwxr-xr-x 24 root root 4096 Feb 11 21:41 latest -> 5
$
```

Each directory in `~/backups` is a backup generation, named after the generation identifier Obnam invents. The `latest` symbolic link points at the latest generation.

After this, you can restore a single file very easily:

```
cp ~/backups/latest/home/tomjon/Documents/iloveyou.txt ~/restored.txt
```

You can copy any files you want from the `~/backups` directory, from any generation, or all of them if you want to. You can look at files directly, before copying them out, too.

```
less ~/backups/2/home/tomjon/Documents/iloveyou.txt
```

This is an easy way to make sure you find the right version instead of just the latest one.

You can't delete anything in `~/backups`. That directory is read-only, and you can't accidentally, or on purpose, delete or modify anything there. This is intentional: the `obnam mount` command is meant to be a safe way for you to look at your backups, not something you need to be careful about.

Once you're done looking at your backups, you can un-mount the repository:

```
fusermount -u ~/backups
```

In addition to doing these things from the command line, you can, of course, use your favourite file manager (graphical or textual) to look at your backed up files. The mounting and un-mounting (depending on your desktop setup) may need to be done on the command line.

6.2 Restoring without FUSE

When `obnam mount` isn't available, you can do restores directly with just Obnam. Use `obnam generations` and `obnam ls` to find the right generation to restore, and then run a command like this:

```
obnam restore --to /tmp/tomjon-restored /home/tomjon/Documents
```

This would restore just the indicated directory. If you don't tell Obnam what to restore, it'll restore everything in the latest generation. You can choose a different generation with `--generation`:

```
obnam restore --to /tmp/tomjon-restored --generation 2
```

Note that you can't restore to a directory that already exists. This is to prevent you from accidentally overwriting your live data with restored files. If you do want replace your live data with restored files, you should restore to a temporary location first, and then move the files to where you want them to be.

The file or directory you want to restore must be specified using an absolute path, i.e., starting from / and not relative to the current directory.

6.3 An actual example of a restoration

I had a corrupted gnus file, and this is how I restored it from backup.

```
obnam --config=/home/foobar/cron/conf/obnam.conf generations>~/cron/upload/obgen.txt
```

This copies all generations for the main obnam backup to obgen.txt, and this is part of that file.

```
1207586 2014-08-25 08:00:43 .. 2014-08-25 08:08:24 (385163 files, 175029819657 bytes)
1208367 2014-08-25 12:00:42 .. 2014-08-25 12:08:31 (385965 files, 175057598863 bytes)
1209313 2014-08-25 16:00:12 .. 2014-08-25 16:07:33 (386537 files, 175076976590 bytes)
1210254 2014-08-25 20:00:15 .. 2014-08-25 20:09:41 (386896 files, 175086483254 bytes)
```

And I decided to restore from generation 1208367.

This is the actual restore command.

```
obnam --config=/home/foobar/cron/conf/obnam.conf --generation=1208367 restore ~/News/rss/nnrss.el
```

This restores 'nnrss.el' to ~/cron/upload/ from where I was able to copy it back to its proper place in ~/News/rss/

Obviously you replace your user-name for foobar.

6.4 Practice makes prestores painless

You should practice doing restores. This makes you trust your backups more, and lets you be calmer if disaster were to strike. (In fancier terms, you should test your disaster recovery plan.)

Do a trial restore of a few files, or all files, until you're sure you know how to do that. Then do it again, from time to time, to be sure your backups still work. It's much less frightening to do a real restore, when data has actually gone missing, if you've done it before.

In extreme cases, particularly if you're an Obnam developer, you perhaps format your hard drive and then do complete restore, just so you know you can. If you're not an Obnam developer, this is perhaps a bit extreme: at least use a separate hard drive instead of your normal one.

Chapter 7

Forgetting old backup generations

Every time you make a backup, your backup repository grows in size. To avoid overflowing all available storage, you need to get rid of some old backups. That's a bit of a dilemma, of course: you make backups in order to not lose data and now you have to do exactly that.

Obnam uses the term "forget" about removing a backup generation. You can specify which generation to remove manually, by generation identifier, or you can have a schedule to forget them automatically.

To forget a specific generation:

```
obnam forget 2
```

(This example assumes you have a configuration file that Obnam finds automatically, and that you don't need to specify things like repository location or encryption on the command line.)

You can forget any individual generation. Thanks to the way Obnam treats every generation as an independent snapshot (except it's not really a full backup every time), you don't have to worry about the distinctions between a full and incremental backup.

Forgetting backups manually is tedious, and you probably want to use a schedule to have Obnam automatically pick the generations to forget. A common type of schedule is something like this:

- keep one backup for each day for the past week
- keep one backup for each week for the past three months

- keep one backup for each month for the past two years
- keep one backup for each year for the past fifty seven years

Obnam uses the `--keep` setting to specify a schedule. The setting for the above schedule would look like this:

```
--keep 7d,15w,24m,57y
```

This isn't an exact match, due to the unfortunate ambiguity of how long a month is in weeks, but it's close enough. The setting "7d" is interpreted as "the last backup of each calendar day for the last seven days on which backups were made". Similarly for the other parts of the schedule. See the "Obnam configuration files and settings" chapter for exact details.

The schedule picks a set of generations to keep. Everything else gets forgotten.

7.1 Choosing a schedule for forgetting generations

The schedule for retiring backup generations is a bit of a guessing game, just like backups in general. If you could reliably tell the future, you'd know all the disasters that threaten your data, and you could backup only the things that would otherwise be lost in the future.

In this reality, alas, you have to guess. You need to think about what risks you're facing (or your data is), and how much backup storage space you're willing to spend on protecting against them.

- Are you afraid of your hard drive suddenly failing in a very spectacular way, such as by catching fire or being stolen? If so, you really only need one very recent backup to cover against that.
- Do you worry about your hard drive, or filesystem, or your application programs, or you yourself, slowly corrupting your data over a longer period of time? How long will it take you to find that out? You need a backup history that lasts longer than it takes for you to detect slow corruption.
- Likewise with accidental deletion of files. How long will it take for you to notice? That's how long the backup history should be, at minimum.

There's other criteria as well. For example, would you like to see, in fifty years, how your files are laid out today? If so, you need a fifty-year-old backup, plus perhaps a backup from each year, if you want to compare how things were each year in between. With increasing storage space and nice de-duplication features, this isn't quite as expensive as it might be.

There is no one schedule that fits everyone's needs and wants. You have to decide for yourself. That's why the default in Obnam is to keep everything forever. It's not Obnam's duty to decide that you should not keep this or that backup generation.

Chapter 8

Verifying backups

It's 9 in the evening. Do you know if your backups work? Do you know when you last made a successful backup of all of your data? Do you know whether you can restore from that backup? If not, how well can you sleep?

You should verify your backups, and do it regularly, not just when you first set up the backup system. Verification means doing whatever you need to do to ensure all of your precious data has been backed up and can be correctly restored from the backups.

The simplest way to do that is to restore all your data, and compare it with your live data, and note any differences. That requires you have enough free disk space to restore everything, but it's almost the only way to be really sure.

It's also a great way to ensure the restoring actually works. If you don't test that, don't expect it'll work when needed.

If you have the disk space to do a complete restore, doing so is a great way to exercise your disaster recovery process in general. Here's one way of doing it:

- On your main computer, do a backup.
- On a second computer, perhaps borrowed for this, restore all your data, without using your main computer at all.
- Start using the restored data as your live data. Do real work, and do all the things you normally do. Pretend your main computer was eaten by your pet shark.
- If you notice something missing, or being corrupt, or being too old, get the files from your main computer, and fix your backup process so that the next time you won't have that problem.

How often should you do that? That, again, depends on how you feel about your data, and how much you trust your backup tools and processes. If it's

really important that you can recover from a disaster, you need to verify more frequently. If data loss is merely inconvenient and not disastrous in a life changing way, you can verify less often.

In addition to restoring data, Obnam provides two other ways to verify your backups:

- `obnam verify` is like `obnam restore`, except it compares the backed up data with live data, and reports any differences. This requires you to trust that Obnam does the verification correctly.
- `obnam mount` lets you access your backed up data as if it were just a directory. You can then use any tool you trust to compare the backed up data with live data. This is very much like doing a restore, since the comparison tool will have to extract all the data and metadata from the backup; it just doesn't write it out.

Both of these approaches have the problem that they compare a backup with live data, and the live data may have changed after the backup was made. You need to verify all differences manually, and if the live data changes frequently, the can be a large number of wrong reports.

Chapter 9

Sharing a repository between multiple clients

Obnam lets you backup several computers to the same repository. Each client is identified by a name, which defaults to the system hostname: the name you get when you run the `hostname` command. You can also set the name explicitly, using the `--client-name` setting in Obnam.

All the clients sharing a repository share the file content data (the chunks), and can de-duplicate across clients. Each client has its own backup generations, and those are fully independent from other clients. You can, for example, forget any generations you want for one client, and it doesn't affect the generations or any backed up data for any other client.

Obnam takes care of locking automatically so you can run Obnam on each client without having to arrange it so that you only run it on one client at a time.

A caveat of sharing a repository is that any client has access to all chunks, and can delete any other client from the repository. This means you should only share a repository amongst clients in the same security domain: all clients should be trusted equally. If one client gets hacked, then the intruder has access to all the data in the repository, and can delete the backups of all the clients using that repository.

To share a repository amongst clients you need to do the following:

- Set a unique name for each client. It needs to be unique within the repository.
- Arrange for each client to have access to the repository.

That's all.

To see what clients are using a repository, use this:

```
obnam clients
```

There is currently no way to remove a client from a repository, unless you're using encryption. This is to be considered a bug in Obnam, and will be fixed at a future time. After that, a time machine will be developed so that this paragraph will have never existed.

Chapter 10

Using encryption

Obnam allows you to encrypt your backups. This chapter discusses why and how to do that.

10.1 You don't admit to being a spy, so isn't encryption unnecessary?

You're not the only one who cares about your data. A variety of governments, corporations, criminals, and overly curious snoopers und lookenpeepers may also be interested. (It's sometimes hard to tell them apart.) They might be interested in it to data in order to find evidence against you, blackmail you, or just curious about what you're talking about with your other friends.

They might be interested in your data from a statistical point of view, and don't particularly care about your specifically. Or they might be interested only in you.

Instead of reading your files and e-mail, or looking at your photos and videos, they might be interested in preventing your access to them, or to destroy your data. They might even want to corrupt your data, perhaps by planting child porn in your photo archive.

You protect your computer as well as you can to prevent these and other bad things from happening. You need to protect your backups with equal care.

If you back up to a USB drive, you should probably make the drive be encrypted. Likewise, if you back up to online storage. There are many forms of encryption, and I'm unqualified to give advice on this, but any of the common, modern ones should suffice except for quite determined attackers.

Instead of, or in addition to, encryption, you could ensure the physical security of your backup storage. Keep the USB drive in a safe, perhaps, or a safe deposit box.

The multiple backups you need to protect yourself against earthquakes, floods, and roving gangs of tricycle-riding clowns, are also useful against attackers. They might corrupt your live data, and the backups at your home, but probably won't be able to touch the USB drive encased in concrete and buried in the ground at a secret place only you know about.

The other side of the coin is that you might want to, or need to, ensure others do have access to your backed up data. For example, if the clown gang kidnaps you, your spouse might need access to you backups to be able to contact your MI6 handler to ask them to rescue you. Arranging safe access to (some) backups is an interesting problem to which there are various solutions. You could give your spouse the encryption passphrase, or give the passphrase to a trusted friend or your lawyer. You could also use something like libgfshare to escrow encryption keys more safely.

10.2 How Obnam encryption works

An Obnam repository contains several directories, for different types of data.

- A per-client directory for each client, for data that is only relevant to that client, such as the generations to that client.
- A directory for the list of clients.
- A directory for all the chunks of file content data, plus additional directories used for de-duplicating chunks.

The per-client directory is encrypted so that only that client can access it. This means that only the client itself can see its generations, and the files in each generation.

The shared directories (client list, chunks) is encrypted so that all clients can use them. This allows clients to share chunks, so that de-duplication works across all clients.

This encryption scheme assumes that all clients sharing a repository trust each other, and that it's OK for them to be able to read all the chunk data they want. The encryption does not protect siblings from reading each others e-mail from the backup repository, for example, but it does protect them against their parents, if the parents don't have a suitable encryption key.

In addition to the encryption keys for a client you can add additional keys. These keys can also access the backup repository. For example, the parents' key might

be added to the repository so that if need be, they could restore any child's data, even if the child had lost their own encryption key.

In a corporate setting, the a backup administrator key might be added so that the administrator can, for example, verify the integrity of the repository, or to access data of an employee who has won the lottery and isn't currently available due to bad Internet access to the Moon.

Such additional keys can be added either for any one client, or to all clients.

10.3 Setting up Obnam to use encryption

Obnam uses PGP keys, specifically the GNU Privacy Guard (GnuPG, gpg) implementation of them. To use encrypted backups, you need to first create a PGP key pair for yourself. See the GnuPG documentation for instructions.

Once you have a working GnuPG setup and a key pair (consisting of a public key and a secret key), you need to find the key identifier for them. Run the following command and pick your key from the list.

```
gpg --list-keys
```

The output will look something like this:

```
pub   4096R/5E8511F9 2009-07-22
uid                  Lars Wirzenius <liw@liw.fi>
sub   2048R/9BE35AE6 2011-08-05
```

That's the output for one key; there may be many keys. The key identifier is on the line staring with `pub`, in the second column after the slash. Above, it's 5E8511F9.

In the rest of the examples in this chapter, we'll assume your key identifier is CAFEFACE.

To set up encryption, use the `--encrypt-with` setting:

```
[config]
encrypt-with = CAFEFACE
```

That's all.

Note that a repository should be fully encrypted or not encrypted at all, and that you can't switch afterwards. If you change your mind about whether to use encryption at all, you'll need to start a new repository. All clients sharing a repository need to be using encryption, or else none of them may use encryption.

If you mix encryption or cleartext backups, the error messages may prove to be confusing.

Obnam will automatically encrypt all the files it writes to the backup repository, and de-crypt them when needed. As long as you only have one encryption key for each client, and don't add additional keys, Obnam will take care of adding the right keys to the right places automatically.

10.4 Checking if a repository uses encryption

There is no direct way with Obnam to check if a repository uses encryption. However, you can check that manually: if your repository contains the file `clientlist/key`, the repository is encrypted.

10.5 FIXME: Managing encryption keys in a repository

This section discusses how to manage encryption keys in a repository: how to add additional keys for each toplevel, and how to change keys for a client. It also shows how to check what keys are being used, and what access each key has.

Chapter 11

Other stuff

This chapter discusses topics that do not warrant a chapter of their own, such as compressing backups and running Obnam from cron.

11.1 k4dirstat cache files

k4dirstat is a utility for visualising the disk space used by a directory tree. Obnam's `kdirstat` command can be used to produce a listing of the contents of a generation in a format which can be read by k4dirstat using `File`, `Read Cache File` from the k4dirstat menu. e.g.

```
$ obnam kdirstat --client CLIENT --generation GENID > CLIENT.kdirstat.cache
$ gzip -v9 CLIENT.kdirstat.cache # OPTIONAL
```

`CLIENT.kdirstat.cache[.gz]` can now be read by `k4dirstat`.

Chapter 12

Case studies

This chapter goes through, in some detail, some typical use cases for backups. For case, it discusses the data being backed up, and explains choices of backup strategy, storage, etc. Some case studies:

- Single laptop user, typical data of documents, photos, music, backing up to a USB hard drive.
- A VPS or similar server, with web pages, e-mail, and maybe other data, backed up to another server.
- A small company with a number of laptops and desktops, a local file server, a rented co-lo server, backing up to a rented server in a co-lo and to a rotated set of large USB drives.
- Restoring from a complete disaster, where all local computers and storage media are destroyed, but there is an off-site backup that is intact.

Chapter 13

Troubleshooting

This chapter discusses how to debug problems with Obnam. It covers things such as log files, various levels of logging and tracing, and common problems with Obnam use. It also explains what things go where in an Obnam backup repository.

13.1 Turning on full logging

Obnam can write a log file. There are several options controlling that. Knowing these can help get out the most information when there's a problem that needs to be investigated.

- `--log=obnam.log` tells Obnam where to log. The log is a simple text file.
- `--log-level=debug` tells Obnam to log at the most detailed level. The default level is `info`, which excludes most debug information.
- `--trace=obnamlib --trace=larch` tells Obnam to log additional debug information. The two arguments match all filenames in Obnam and the Larch library Obnam uses. This additional information is mostly useful to someone who can read and understand the program source code.

Note that these settings can make log files be quite large, in the order of tens of megabytes. The size depends on how many files and how much data your live data has.

13.2 Reporting problems ("bugs")

If you have a problem with Obnam, and you want to report it (please do!), including the following information is helpful and makes it easier to figure out

what the problem is.

- You should report problems to the `obnam-support@obnam.org` mailing list. This is a publicly archived mailing list where various people help others use Obnam.

If you respond to messages on `obnam-support`, **always** keep the list in the cc list. This means others will see the response, and there's a chance that they can help you better than the particular person you're responding to. Also, the archived discussion may be helpful to later readers, perhaps years afterwards.

- What is the problem? What did you try to achieve? What actually happened?
- The version of Obnam and Larch you're using, and how you installed it.
  - On Debian, run `dpkg -l obnam python-larch` on the command line and include the output.
- The exact command line you used. Copy-paste it instead of typing it again into the mail. Sometimes the problem can be hidden if you don't copy the command line exactly. Also, copying by typing is boring, and we should avoid boring things in life.
- If there's an error message, copy-paste that into the mail.
- The output of `obnam --dump-config`, which includes the full configuration. Include it as an attachment to your mail to `obnam-support`. If you have some secret information, such as filenames or hostnames, you can replace those with XXXX.
- If you can reproduce the problem while running with `--log-level=debug`, `--log=obnam.log` and `--trace=obnamlib --trace=larch` options, include a suitable amount from the end of the log file. The suitable amount may depend on the situation, but if you give the last two hundred lines, and it's not enough, we'll ask for more. Again, feel free to replace any sensitive filenames, etc, with XXXX.
- The output of the `env` command, in the same terminal window in which you ran Obnam. (Again, as an attachment.)
- If your bug is about performance, please run Obnam under profiling, and attach the profiling file. To run Obnam under profiling, install the Python profile (`python-profiler` package in Debian/Ubuntu), and set the `OBNAM_PROFILE` environment variable to the name of the file with the profiling output (that's the file you should send by mail). For example: `OBNAM_PROFILE=obnam.prof obnam backup` would run the backup under the profiler, and write the result to `obnam.prof`.

Thank you for your help in making Obnam better.

Chapter 14

Obnam configuration files and settings

This chapter discusses Obnam configuration files: where they are, what they contain, and how they are used.

14.1 Where is my configuration?

Obnam looks for its configuration files in a number of places:

- `/etc/obnam.conf`
- `/etc/obnam/*.conf`
- `~/.obnam.conf`
- `~/.config/obnam/*.conf`

Note that in `/etc/obnam` and `~/.config/obnam`, all files that have a `.conf` suffix are loaded, in "asciibetical" order, which is like alphabetical, but based on character codes rather than what humans understand, but unlike alphabetical isn't dependent on the language being used.

Any files in the list above may or may not exist. If it exists, it is read, and then the next file is read. A setting in one file can be overridden by a later file, if it is set there as well. For example, `/etc/obnam.conf` might set `log-level` to `INFO`, but `~/.obnam.conf` may then set it to `DEBUG`, if a user wants more detailed log files.

The Obnam configuration files in `/etc` apply to everyone who runs Obnam on that machine. This is important: they are not just for when `root` runs Obnam.

If you want to have several Obnam configurations, for example for different backup repositories, you need to name or place the files so they aren't on the list above. For example:

- `/etc/obnam/system-backup.profile`
- `~/.config/obnam/online.profile`
- `~/.config/obnam/usbdrive.profile`

You would then need to specify that file for Obnam to use it:

```
obnam --config ~/.config/obnam/usbdrive.profile`
```

If you want to not be affected by any configuration files, except the ones you specify explicitly, you need to also use the `--no-default-config` option:

```
obnam --no-default-config --config ~/.obnam-is-fun.conf
```

Command line options override values from configuration files.

14.2 Configuration file syntax

Obnam configuration files use the INI file syntax, specifically the variant implemented by the Python ConfigParser library. They look like this:

```
[config]
log-level = debug
log = /var/log/obnam.log
encrypt-with = CAFEBEEF
root = /
one-file-system = yes
```

Names of configuration variables are the same as the corresponding command line options. If `--foo` is the option, then the variable in the file is `foo`. Any command line option `--foo=bar` can be used in a configuration file as `foo = bar`. There's are exceptions to this (`--no-default-config`, `--config`, `--help`, and a few others), but they're all things you wouldn't put in a configuration file anyway.

Every option, or setting, has a type. Mostly this doesn't matter, unless you give it a value that isn't suitable. The two important exceptions to this are:

- Boolean or yes/no or on/off settings. For example, `--exclude-caches` is a setting that is either turned on (when the option is used) or off (when it's not used). For every Boolean setting `--foo`, there is an option `--no-foo`. In a configuration files, `foo` is turned on by setting it to `yes` or `true`, and off by setting it to `no` or `false`.
- Some settings can be lists of values, such as `--exclude`. You can use `--exclude` as many times as you want, each time a new exclusion pattern is added, rather than replacing the previous patterns. In a configuration file, you would write all the values at once, separated by commas and optional spaces: for example, `exclude = foo, bar, baz`. In a configuration file, the previous list of values is replaced entirely rather than added to.

For a more detailed explanation of Obnam configuration file syntax, see the **cliapp**(5) manual page on your system, or cliapp man page on the WWW.

14.3 Checking what my configuration is

Obnam can read configuration files from a number of places, and it can be tricky to figure out what the actual configuration is. The `--dump-config` option helps here.

```
obnam --config ~/.obnam.fun --exclude-caches --dump-config
```

The option will tell Obnam to write out (to the standard output) a configuration file that captures every setting, and reporting the value that it would have if `--dump-config` weren't used.

This is a good way to see what the current settings are and also as a starting point if you want to make a configuration file from scratch.

14.4 Finding out all the configuration settings

This manual does not yet have a list of all the settings, and their explanation. Obnam provides built-in help (run `obnam --help`) and a manual page automatically generated from the built-in help (run `man obnam` or see obnam man page). Some day, this chapter will include an automatically generated section that explains each setting. Until then, you're free to point fingers at Obnam's author and giggle at his laziness.

Chapter 15

The backup repository internals

This chapter describes what the Obnam backup repository looks like. Unless you're interested in that, you can skip that entirely.

For now, look at the Obnam website at http://obnam.org/development/.

15.1 Repository file permissions

Obnam sets the permissions of all files it creates in the repository such that only the owner of the files can read or write them. (Technically, 0600 for files, 0700 for directories.)

This is to prevent backups from leaking because someone else has read access to the repository. There is no setting in Obnam to control this.

Chapter 16

Performance tuning

This chapter discusses ways to tune Obnam performance for various situations. It covers the various options that can affect CPU and memory consumption, as well as ways to experiment to find a good set of values.

See http://obnam.org/faq/tuning/ for a start.

16.1 Running Obnam under the Python profiler

A **profiler** is a program that measures where another program spends its time. This can be very useful for finding out why the other program is slow.

Obnam can easily be run under the Python profiler. You need to have the profiler installed. Check with your operating system or Python installation how to achieve that. To see if you have it installed, run the following command on the command line:

```
python -c 'import cProfile'
```

If this outputs nothing, all is well. If it outputs an error such as the following, you have not got the profiler installed:

```
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ImportError: No module named cProfiler
```

Once you have the profiler installed, run Obnam like this:

```
OBNAM_PROFILE=backup.prof obnam backup
```

This will cause the profiling data to be written to the file `backup.prof`. You can do this for any Obnam command, and write it to any file.

The profiling data is in binary form. Obnam comes with a little helper program to transform it to a human-readable form:

```
obnam-viewprof backup.prof | less
```

If you run the above command, you'll see that the humans to whom this is readable are programmers and circus clowns. If you can understand the output, great! If not, it's still useful to send that to the Obnam developers to report a performance problem.

Chapter 17

Participating in Obnam development

The Obnam project is quite small, as far as software projects go. There is one main developer, plus a few others who sometimes help out. It would be nice to have more people involved, and this chapter is an introduction for that.

It is a common misunderstanding that only code matters in a software project. On the contrary, without a number of other things, code is useless, particularly so in a free software project, including Obnam. Examples of necessary things in almost any serious software project:

- writing and updating documentation, which includes manuals and websites
- translating documentation, and the software's user interface
- providing support to users with questions or problems
- reporting actionable bugs
- processing bug reports: asking for clarifications, reproducing the reported problem, finding the cause of the problem, and developing a suitable fix
- porting the software to various platforms, including different operating systems, different versions of said operating systems, different versions of the languages and libraries the software uses, different hardware, etc
- quality assurance: developing and performing manual and automated tests and benchmarks, and analysing results
- hosting and operating web sites, mailing lists, IRC channels, other communication channels
- handling project governance, which includes dealing with conflicts between people
- managing the project in general, including making sure things don't stall
- finally, writing the code itself, which is a necessary, but not sufficient part of having a project that people other than the developers can use it

This list is insufficient; additions are welcome. See the rest of this chapter for suggestions on how to contribute to the list.

17.1 Helping support users

Perhaps the easiest way to participate in the project is to help support other users of the software. This is easy, and doesn't necessarily require more than being able to use the software oneself. Yet it is quite valuable, as it frees others from doing that. Even with the highest quality, easiest to use software, there's always some need for user support:

- Code can be wrong, and a user may experience this. Analysing the situation and isolating the bug is an important part of the software development process.
- Documentation can be wrong, or out of date, or written in anticipation of a feature that doesn't exist yet.
- Some people have misunderstandings, due to whatever reason, which leads them to have problems when using the software. Figuring out what the actual problem and its cause are can be a time consuming process, but often does not require any special skills, except for patience and a willingness to ask a lot of questions.

In the Obnam project, the best way to help out with this is to subscribe to the `obnam-support@obnam.org` mailing list or join the `#obnam` (irc.oftc.net) IRC channel, and start answering questions.

It's OK to not be an expert. Helping others is a great way to learn. If you make it clear you're not an expert, but are trying to help anyway, usually makes others appreciate your help even more.

Some suggestions on doing support work:

- Try to understand what the person needing help is actually trying to achieve, rather than answering their literal question. Better yet, do both.
- You don't need to have the solution to respond. A quick, but incomplete answer that nevertheless moves the discussion forward is helpful. Even if you don't know the correct answer, it's good to ask a question that results in the person needing help providing more information, or finding the solution themselves, or inspires someone else to discover the solution,
- Always be helpful and polite. Never respond with things such as "read the fine manual" (or RTFM for short). It's OK to say that the answer is in the manual, but then provide a link, and possibly also a quote.

- People who need help are often frustrated, and sometimes desperate, because they've tried and tried to solve the problem on their own, but have failed. This can leak through their messages. Ignore it, unless they actually become impolite, at which point its probably best to escalate the situation. Avoid getting into a quarrel about who's right or who said what and what did they mean by it.

- It's better to not respond at all, than respond while irritated, annoyed, or angry. It's more important for the project to maintain a polite and helpful atmosphere in the long run than to solve any current technical problem.

In short, if you do your best to be polite, friendly, and helpful, go ahead and respond.

17.2 Writing and updating documentation

The project has various kinds of documentation.

- The `obnam.org` website.
- The manual page.
- The manual (which is what you're reading now).
- Various blog posts around the web.

Writing documentation is fairly easy. Updating it takes a bit more effort, since it requires reviewing existing documentation to make sure it's up to date. The main goals of Obnam documentation are:

- Accuracy.
- Clarity.
- Completeness.
- A bit of dry humour in places.

Any help you can give here is most welcome.

- Read through existing documentation.
- If you find anything that's wrong, inaccurate, incomplete, missing, or unclear, send a note to the developer mailing list.
- If you can include a new wording, all the better. It's not required.
- If you can provide an actual patch, perfect, since it makes it easiest to incorporate your suggestion. Again, it's not required.

You don't need to be a good writer. As part of the process, others will review what you send, and will point out anything they feel can be improved. For example, suppose you notice that a paragraph in this manual is unclear, but you don't know what it actually should say. If you send a mail saying this, others can then come up with a better wording.

17.3 Translating documentation

The Obnam manual and manual page are written in English, and have been translated to German. More languages are most welcome.

The author of this manual is not particularly familiar with the process of translation, and so wishes someone else would fill in this section.

The Obnam user interface is not currently translatable, and making it so will require code changes. Helping make those code changes would be nice.

17.4 Developing the code

Assuming you already know how to program, it's fairly straightforward to work on the Obnam code base. At least it's meant to be so: if you have trouble, please ask and point out what's unclear or wrong.

Check out the source from the git server, and read the `README` file for details on how to get started, and how to run the automated test suite, and how to send patches. See the website for some development documentation, including explanations of the on-disk data structures.

Code changes that are not very trivial should be sent in a form that can be handled by git. This can be actual patches sent to the mailing list, or a URL from which changes can be merged.

17.5 Project governance

The Obnam project has a very informal form of governance: the founder of the project, Lars Wirzenius, has all the power, and everyone else has no power. As the project grows, this will change.

If there's a social problem somewhere, for example someone is misbehaving, it's best to report it to Lars directly. If Lars is the problem, it's best to call him out directly.

Chapter 18

Appendix: Error messages

This appendix lists all Obnam error messages and their explanations. It is possible you'll see other error messages while running Obnam. These are not listed here, as Obnam doesn't know about them.

The errors are listed twice: briefly, in order of their unique error, and then more fully, in alphabetical order.

18.1 By error code

- `R018FCX ToplevelIsFileError`
- `R01F56X RepositorySettingMissingError`
- `R02C17X HardlinkError`
- `R0B15DX RepositoryGenerationDoesNotExist`
- `R0BE94X RepositoryClientNotLocked`
- `R0C79EX GpgError`
- `R0F22CX URLSchemeAlreadyRegisteredError`
- `R0FC21X SetMetadataError`
- `R169C6X MissingFilterError`
- `R173AEX NoFilterTagError`
- `R1A025X RepositoryClientKeyNotAllowed`
- `R1CA00X ClientDoesNotExistError`
- `R22E66X SizeSyntaxError`
- `R24424X RepositoryClientDoesNotExist`
- `R283A6X UnitNameError`
- `R2FA37X WrongNumberOfGenerationSettingsError`
- `R338F2X BackupRootMissingError`
- `R3B42AX WrongNumberOfGenerationsForVerify`

- R3E151X RepositoryFileDoesNotExistInGeneration
- R3E1C1X RestoreTargetNotEmpty
- R41CE6X RepositoryClientAlreadyExists
- R43272X RepositoryChunkDoesNotExist
- R45B50X DuplicatePeriodError
- R47416X WrongHostKeyError
- R4C3BCX BackupErrors
- R57207X RepositoryClientGenerationUnfinished
- R5914DX InvalidPortError
- R5F98AX NoHostKeyError
- R681AEX LockFail
- R6A098X RepositoryGenerationKeyNotAllowed
- R6C1C8X RepositoryClientListNotLocked
- R6EAF2X RepositoryClientLockingFailed
- R7137EX BagIdNotSetError
- R79699X RepositoryFileKeyNotAllowed
- R79ED6X BackupRootDoesNotExist
- R7B8D0X FileNotFoundError
- R826A1X UnknownVFSError
- R8AAC1X NoHostKeyOfWantedTypeError
- R8F974X RepositoryChunkIndexesLockingFailed
- R91CA1X ShowFirstGenerationError
- R9808DX ForgetPolicySyntaxError
- RA4F35X RootIsNotADirectory
- RA5942X WrongNumberOfGenerationsForDiffError
- RA7D64X UnknownRepositoryFormatWanted
- RA881CX RepositoryChunkContentNotInIndexes
- RA920EX NotARepository
- RABC26X FuseModuleNotFoundError
- RB1048X RepositoryClientListLockingFailed
- RB4324X GAImmutableError
- RB8E98X WrongURLSchemeError
- RB927BX SeparatorError
- RBF6DDX RepositoryAccessError
- RCB0CAX KeyAuthenticationError
- RCE08AX ObnamIOError
- RCEF5CX MallocError
- RD5FA4X ObnamSystemError
- RD6259X RestoreErrors
- RDF30DX Fail
- RE187FX RepositoryChunkIndexesNotLocked
- REFB32X RepositoryClientHasNoGenerations

- `RF4EFDX UnknownRepositoryFormat`

18.2 By name

`BackupErrors (R4C3BCX)` There were errors during the backup

`BackupRootDoesNotExist (R79ED6X)` Backup root does not exist or is not a directory: {root}

`BackupRootMissingError (R338F2X)` No backup roots specified

`BagIdNotSetError (R7137EX)` Bag id not set: cannot append a blob (programming error)

`ClientDoesNotExistError (R1CA00X)` Client {client} does not exist in repository {repo}

`DuplicatePeriodError (R45B50X)` Forget policy may not duplicate period ({period}): {policy}

`Fail (RDF30DX)` {filename}: {reason}

`FileNotFoundError (R7B8D0X)` FUSE: File not found: {filename}

`ForgetPolicySyntaxError (R9808DX)` Forget policy syntax error: {policy}

`FuseModuleNotFoundError (RABC26X)` Failed to load module “fuse”, try installing python-fuse

`GAImmutableError (RB4324X)` Attempt to modify an immutable GADirectory

`GpgError (R0C79EX)` gpg failed with exit code {returncode}: {stderr}

`HardlinkError (R02C17X)` Cannot hardlink on SFTP; sorry

This is due to a limitation in the Python paramiko library that Obnam uses for SSH/SFTP access.

`InvalidPortError (R5914DX)` Invalid port number {port} in {url}: {error}

`KeyAuthenticationError (RCB0CAX)` Can’t authenticate to SSH server using key

`LockFail (R681AEX)` Couldn’t create lock {lock_name}: {reason}

`MallocError (RCEF5CX)` malloc out of memory while calling {function}

`MissingFilterError (R169C6X)` Unknown filter tag: {tagname}

`NoFilterTagError (R173AEX)` No filter tag found

`NoHostKeyError (R5F98AX)` No known host key for {hostname}

NoHostKeyOfWantedTypeError (R8AAC1X) No known type {key_type} host key for {hostname}

NotARepository (RA920EX) {url} does not seem to be an Obnam repository

ObnamIOError (RCE08AX) I/O error: {filename}: {errno}: {strerror}

ObnamSystemError (RD5FA4X) System error: {filename}: {errno}: {strerror}

RepositoryAccessError (RBF6DDX) Repository does not exist or cannot be accessed: {error}

RepositoryChunkContentNotInIndexes (RA881CX) Repository chunk indexes do not contain content

RepositoryChunkDoesNotExist (R43272X) Repository doesn't contain chunk {chunk_id}. It is expected at {filename}

RepositoryChunkIndexesLockingFailed (R8F974X) Repository chunk indexes are already locked

RepositoryChunkIndexesNotLocked (RE187FX) Repository chunk indexes are not locked

RepositoryClientAlreadyExists (R41CE6X) Repository client {client_name} already exists

RepositoryClientDoesNotExist (R24424X) Repository client {client_name} does not exist

RepositoryClientGenerationUnfinished (R57207X) Cannot start new generation for {client_name}: previous one is not finished yet (programming error)

RepositoryClientHasNoGenerations (REFB32X) Client {client_name} has no generations

RepositoryClientKeyNotAllowed (R1A025X) Client {client_name} uses repository format {format} which does not allow the key {key_name} to be use for clients

RepositoryClientListLockingFailed (RB1048X) Repository client list could not be locked

RepositoryClientListNotLocked (R6C1C8X) Repository client list is not locked

RepositoryClientLockingFailed (R6EAF2X) Repository client {client_name} could not be locked

RepositoryClientNotLocked (R0BE94X) Repository client {client_name} is not locked

RepositoryFileDoesNotExistInGeneration (R3E151X) Client {client_name}, generation {genspec} does not have file {filename}

RepositoryFileKeyNotAllowed (R79699X) Client {client_name} uses repository format {format} which does not allow the key {key_name} to be use for files

RepositoryGenerationDoesNotExist (R0B15DX) Cannot find requested generation {gen_id!r} for client {client_name}

RepositoryGenerationKeyNotAllowed (R6A098X) Client {client_name} uses repository format {format} which does not allow the key {key_name} to be used for generations

RepositorySettingMissingError (R01F56X) No –repository setting. You need to specify it on the command line or a configuration file

RestoreErrors (RD6259X) There were errors when restoring

See previous error messages for details.

RestoreTargetNotEmpty (R3E1C1X) The restore –to directory ({to}) is not empty.

RootIsNotADirectory (RA4F35X) {baseurl} is not a directory, but a VFS root must be a directory

SeparatorError (RB927BX) Forget policy must have rules separated by commas, see position {position}: {policy}

SetMetadataError (R0FC21X) {filename}: Couldn't set metadata {metadata}: {errno}: {strerror}

ShowFirstGenerationError (R91CA1X) Can't show first generation. Use 'obnam ls' instead

SizeSyntaxError (R22E66X) "{size}" is not a valid size

ToplevelIsFileError (R018FCX) File at repository root: {filename}

URLSchemeAlreadyRegisteredError (R0F22CX) VFS URL scheme {scheme} already registered

UnitNameError (R283A6X) "{unit}" is not a valid unit

UnknownRepositoryFormat (RF4EFDX) Unknown format {format} at {url}

UnknownRepositoryFormatWanted (RA7D64X) Unknown format {format} requested

UnknownVFSError (R826A1X) Unknown VFS type: {url}

`WrongHostKeyError (R47416X)` SSH server {hostname} offered wrong public key

Note that this may due to an obsolete host key in your "known hosts" file. If so, use "ssh-key -R" to remove it. However, it can also be a sign that someone is trying to hijack your connection to your server, and you should be careful.

`WrongNumberOfGenerationSettingsError (R2FA37X)` The restore command wants exactly one generation option

`WrongNumberOfGenerationsForDiffError (RA5942X)` Need one or two generations

`WrongNumberOfGenerationsForVerify (R3B42AX)` verify must be given exactly one generation

`WrongURLSchemeError (RB8E98X)` SftpFS used with non-sftp URL: {url}

Chapter 19

SEE ALSO

This chapter gives pointers to more information about Obnam, backups, and related things. For the time being, this is a very short list, but suggestions for things to add to it are very much welcome.

- Obnam home page: http://obnam.org.
  - There are short tutorials, download links, an FAQ, contact information, etc, here.
- Lars Wirzenius, interesting blog tags.
  - http://blog.liw.fi/tag/backups/
  - http://blog.liw.fi/tag/obnam/
- Cache directory tag standard: http://www.bford.info/cachedir/
  - http://liw.fi/cachedir/ is a utility to manage the tag files

Chapter 20

Legal stuff

This entire work is covered by the GNU General Public License, version 3 or later.

> Copyright 2010-2013 Lars Wirzenius
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.

A copy of the GPL is included in the file `COPYING` in the source tree, and can be found at the URL above.

This manual (all the contents of the `manual` subdirectory in the source tree) is additionally licensed under a Creative Commons Attribution 4.0 International License. You can choose whether to use the GPL or the CC license for the manual.

A copy of the Creative Commons license is included in the file `CC-BY-SA-4.0.txt` in the source tree, and can be viewed online at http://creativecommons.org/licenses/by-sa/4.0/legalcode.

Chapter 21

Supporting Obnam development

Obnam is free software: you get full access to the source code, you can modify the software as you wish, and you can distribute copies of the software in its original or modified form. It is also free of charge.

One of the goals of Obnam is to make sure everyone has access to nice backup software, and are not beholden to anyone else for that software. You can use Obnam, and store your backups anywhere that suits you, and the Obnam developers have no say in that.

However, Obnam development requires some resources. Obnam is primarily developed by Lars Wirzenius, its original author (hi!), in his free time. If you would like to help support Obnam development, here's a list of things you could do:

- Send fixes and improvements, either to code or documentation.
- Donate something to the author. See http://obnam.org/donate/ for suggestions.
- Hire the author to do some Obnam development. Contact him privately by e-mail (liw@liw.fi).

Note that any of these are optional. If you like Obnam, and are happy just using it, that's completely OK.