Borg Documentation

Release 1.0.7

see "AUTHORS" file

2016-08-19

Contents

CHAPTER 1

What is BorgBackup?

BorgBackup (short: Borg) is a deduplicating backup program. Optionally, it supports compression and authenticated encryption.

The main goal of Borg is to provide an efficient and secure way to backup data. The data deduplication technique used makes Borg suitable for daily backups since only changes are stored. The authenticated encryption technique makes it suitable for backups to not fully trusted targets.

See the installation manual or, if you have already downloaded Borg, `docs/installation.rst` to get started with Borg.

1.1 Main features

Space efficient storage Deduplication based on content-defined chunking is used to reduce the number of bytes stored: each file is split into a number of variable length chunks and only chunks that have never been seen before are added to the repository.

To deduplicate, all the chunks in the same repository are considered, no matter whether they come from different machines, from previous backups, from the same backup or even from the same single file.

Compared to other deduplication approaches, this method does NOT depend on:

- file/directory names staying the same: So you can move your stuff around without killing the deduplication, even between machines sharing a repo.
- complete files or time stamps staying the same: If a big file changes a little, only a few new chunks need to be stored - this is great for VMs or raw disks.
- The absolute position of a data chunk inside a file: Stuff may get shifted and will still be found by the deduplication algorithm.

Speed

- performance critical code (chunking, compression, encryption) is implemented in C/Cython
- local caching of files/chunks index data
- quick detection of unmodified files

Data encryption All data can be protected using 256-bit AES encryption, data integrity and authenticity is verified using HMAC-SHA256. Data is encrypted clientside.

Compression All data can be compressed by lz4 (super fast, low compression), zlib (medium speed and compression) or lzma (low speed, high compression).

Off-site backups Borg can store data on any remote host accessible over SSH. If Borg is installed on the remote host, big performance gains can be achieved compared to using a network filesystem (sshfs, nfs, ...).

Backups mountable as filesystems Backup archives are mountable as userspace filesystems for easy interactive backup examination and restores (e.g. by using a regular file manager).

Easy installation on multiple platforms We offer single-file binaries that do not require installing anything - you can just run them on these platforms:

- Linux
- Mac OS X
- FreeBSD
- OpenBSD and NetBSD (no xattrs/ACLs support or binaries yet)
- Cygwin (not supported, no binaries yet)

Free and Open Source Software

- security and functionality can be audited independently
- licensed under the BSD (3-clause) license

1.2 Easy to use

Initialize a new backup repository and create a backup archive:

```
$ borg init /path/to/repo
$ borg create /path/to/repo::Saturday1 ~/Documents
```

Now doing another backup, just to show off the great deduplication:

```
$ borg create -v --stats /path/to/repo::Saturday2 ~/Documents
------------------------------------------------------------------------------
Archive name: Saturday2
Archive fingerprint: 622b7c53c...
Time (start): Sat, 2016-02-27 14:48:13
Time (end):   Sat, 2016-02-27 14:48:14
Duration: 0.88 seconds
Number of files: 163
------------------------------------------------------------------------------
               Original size      Compressed size    Deduplicated size
This archive:        6.85 MB              6.85 MB             30.79 kB  <-- !
All archives:       13.69 MB             13.71 MB              6.88 MB

               Unique chunks         Total chunks
Chunk index:             167                  330
------------------------------------------------------------------------------
```

For a graphical frontend refer to our complementary project BorgWeb.

CHAPTER 2

Links

- Main Web Site
- Releases, PyPI packages and ChangeLog
- GitHub, Issue Tracker and Bounties & Fundraisers
- Web-Chat (IRC) and Mailing List
- License

2.1 Notes

Borg is a fork of Attic and maintained by "The Borg collective".

2.1.1 Differences between Attic and Borg

Here's a (incomplete) list of some major changes:

- more open, faster paced development (see issue #1)
- lots of attic issues fixed (see issue #5)
- less chunk management overhead (less memory and disk usage for chunks index)
- faster remote cache resync (useful when backing up multiple machines into same repo)
- compression: no, lz4, zlib or lzma compression, adjustable compression levels
- repokey replaces problematic passphrase mode (you can't change the passphrase nor the pbkdf2 iteration count in "passphrase" mode)
- simple sparse file support, great for virtual machine disk files
- can read special files (e.g. block devices) or from stdin, write to stdout
- mkdir-based locking is more compatible than attic's posix locking
- uses fadvise to not spoil / blow up the fs cache
- better error messages / exception handling
- better logging, screen output, progress indication
- tested on misc. Linux systems, 32 and 64bit, FreeBSD, OpenBSD, NetBSD, Mac OS X

Please read the ChangeLog (or `docs/changes.rst` in the source distribution) for more information.

BORG IS NOT COMPATIBLE WITH ORIGINAL ATTIC (but there is a one-way conversion).

EXPECT THAT WE WILL BREAK COMPATIBILITY REPEATEDLY WHEN MAJOR RELEASE NUMBER CHANGES (like when going from 0.x.y to 1.0.0 or from 1.x.y to 2.0.0).

NOT RELEASED DEVELOPMENT VERSIONS HAVE UNKNOWN COMPATIBILITY PROPERTIES.

THIS IS SOFTWARE IN DEVELOPMENT, DECIDE YOURSELF WHETHER IT FITS YOUR NEEDS.

Borg is distributed under a 3-clause BSD license, see License for the complete license.

Installation

There are different ways to install Borg:

- *Distribution Package* - easy and fast if a package is available from your distribution.
- *Standalone Binary* - easy and fast, we provide a ready-to-use binary file that comes bundled with all dependencies.
- *From Source*, either:
  - *Using pip* - installing a source package with pip needs more installation steps and requires all dependencies with development headers and a compiler.
  - *Using git* - for developers and power users who want to have the latest code or use revision control (each release is tagged).

Distribution Package

Some distributions might offer a ready-to-use `borgbackup` package which can be installed with the package manager. As Borg is still a young project, such a package might be not available for your system yet.

Distribution	Source	Command
Arch Linux	[community]	`pacman -S borg`
Debian	stretch, unstable/sid	`apt install borgbackup`
NetBSD	pkgsrc	`pkg_add py-borgbackup`
NixOS	.nix file	N/A
OS X	Brew cask	`brew cask install borgbackup`
Ubuntu	Xenial 16.04, Wily 15.10 (backport PPA)	`apt install borgbackup`
Ubuntu	Trusty 14.04 (backport PPA)	`apt install borgbackup`

Please ask package maintainers to build a package or, if you can package / submit it yourself, please help us with that! See #105 on github to followup on packaging efforts.

If a package is available, it might be interesting to check its version and compare that to our latest release and review the Changelog.

Standalone Binary

Borg binaries (generated with pyinstaller) are available on the releases page for the following platforms:

- **Linux**: glibc >= 2.13 (ok for most supported Linux releases). Maybe older glibc versions also work, if they are compatible to 2.13.
- **Mac OS X**: 10.10 (does not work with older OS X releases)

- **FreeBSD**: 10.2 (unknown whether it works for older releases)

To install such a binary, just drop it into a directory in your `PATH`, make borg readable and executable for its users and then you can run `borg`:

```
sudo cp borg-linux64 /usr/local/bin/borg
sudo chown root:root /usr/local/bin/borg
sudo chmod 755 /usr/local/bin/borg
```

Note that the binary uses /tmp to unpack Borg with all dependencies. It will fail if /tmp has not enough free space or is mounted with the `noexec` option. You can change the temporary directory by setting the `TEMP` environment variable before running Borg.

If a new version is released, you will have to manually download it and replace the old version using the same steps as shown above.

Features & platforms

Besides regular file and directory structures, Borg can preserve

- Hardlinks (considering all files in the same archive)
- Symlinks (stored as symlink, the symlink is not followed)
- Special files:
  - Character and block device files (restored via mknod)
  - FIFOs ("named pipes")
  - Special file *contents* can be backed up in `--read-special` mode. By default the metadata to create them with mknod(2), mkfifo(2) etc. is stored.
- Timestamps in nanosecond precision: mtime, atime, ctime
- Permissions:
  - IDs of owning user and owning group
  - Names of owning user and owning group (if the IDs can be resolved)
  - Unix Mode/Permissions (u/g/o permissions, suid, sgid, sticky)

On some platforms additional features are supported:

Platform	ACLs [1]	xattr [5]	Flags [6]
Linux x86	Yes	Yes	No
Linux PowerPC			
Linux ARM			
Mac OS X	Yes	Yes	Yes (all)
FreeBSD	Yes	Yes	
OpenBSD	n/a	n/a	
NetBSD	n/a	No [2]	
Solaris 11	No [3]		n/a
OpenIndiana			
Windows (cygwin)	No [4]	No	No

Some Distributions (e.g. Debian) run additional tests after each release, these are not reflected here.

Other Unix-like operating systems may work as well, but have not been tested at all.

Note that most of the platform-dependent features also depend on the file system. For example, ntfs-3g on Linux isn't able to convey NTFS ACLs.

From Source

Dependencies To install Borg from a source package (including pip), you have to install the following dependencies first:

- Python 3 >= 3.4.0, plus development headers. Even though Python 3 is not the default Python version on most systems, it is usually available as an optional install.
- OpenSSL >= 1.0.0, plus development headers.
- libacl (that pulls in libattr also), both plus development headers.
- liblz4, plus development headers.
- some Python dependencies, pip will automatically install them for you
- optionally, the llfuse Python package is required if you wish to mount an archive as a FUSE filesystem. See setup.py about the version requirements.

If you have troubles finding the right package names, have a look at the distribution specific sections below and also at the Vagrantfile in our repo.

In the following, the steps needed to install the dependencies are listed for a selection of platforms. If your distribution is not covered by these instructions, try to use your package manager to install the dependencies. On FreeBSD, you may need to get a recent enough OpenSSL version from FreeBSD ports.

After you have installed the dependencies, you can proceed with steps outlined under *Using pip*.

Debian / Ubuntu Install the dependencies with development headers:

```
sudo apt-get install python3 python3-dev python3-pip python-virtualenv \
libssl-dev openssl \
libacl1-dev libacl1 \
```

[1] The native access control list mechanism of the OS. This normally limits access to non-native ACLs. For example, NTFS ACLs aren't completely accessible on Linux with ntfs-3g.

[5] extended attributes; key-value pairs attached to a file, mainly used by the OS. This includes resource forks on Mac OS X.

[6] aka *BSD flags*.

[2] Feature request #1332

[3] Feature request #1337

[4] Cygwin tries to map NTFS ACLs to permissions with varying degress of success.

```
liblz4-dev liblz4-1 \
build-essential
sudo apt-get install libfuse-dev fuse pkg-config    # optional, for FUSE support
```

In case you get complaints about permission denied on `/etc/fuse.conf`: on Ubuntu this means your user is not in the `fuse` group. Add yourself to that group, log out and log in again.

Fedora / Korora Install the dependencies with development headers:

```
sudo dnf install python3 python3-devel python3-pip python3-virtualenv
sudo dnf install openssl-devel openssl
sudo dnf install libacl-devel libacl
sudo dnf install lz4-devel
sudo dnf install gcc gcc-c++
sudo dnf install fuse-devel fuse pkgconfig         # optional, for FUSE support
```

Mac OS X Assuming you have installed homebrew, the following steps will install all the dependencies:

```
brew install python3 lz4 openssl
brew install pkg-config                            # optional, for FUSE support
pip3 install virtualenv
```

For FUSE support to mount the backup archives, you need at least version 3.0 of FUSE for OS X, which is available as a pre-release.

FreeBSD Listed below are packages you will need to install Borg, its dependencies, and commands to make fuse work for using the mount command.

```
pkg install -y python3 openssl liblz4 fusefs-libs pkgconf
pkg install -y git
python3.4 -m ensurepip # to install pip for Python3
To use the mount command:
echo 'fuse_load="YES"' >> /boot/loader.conf
echo 'vfs.usermount=1' >> /etc/sysctl.conf
kldload fuse
sysctl vfs.usermount=1
```

Cygwin

Note: Running under Cygwin is experimental and has only been tested with Cygwin (x86-64) v2.5.2.

Use the Cygwin installer to install the dependencies:

```
python3 python3-setuptools
binutils gcc-g++
libopenssl openssl-devel
liblz4_1 liblz4-devel
git make openssh
```

You can then install `pip` and `virtualenv`:

```
easy_install-3.4 pip
pip install virtualenv
```

Using pip Virtualenv can be used to build and install Borg without affecting the system Python or requiring root access. Using a virtual environment is optional, but recommended except for the most simple use cases.

Note: If you install into a virtual environment, you need to **activate** it first (`source borg-env/bin/activate`), before running `borg`. Alternatively, symlink `borg-env/bin/borg` into some directory that is in your `PATH` so you can just run `borg`.

This will use `pip` to install the latest release from PyPi:

```
virtualenv --python=python3 borg-env
source borg-env/bin/activate

# install Borg + Python dependencies into virtualenv
pip install borgbackup
# or alternatively (if you want FUSE support):
pip install borgbackup[fuse]
```

To upgrade Borg to a new version later, run the following after activating your virtual environment:

```
pip install -U borgbackup  # or ... borgbackup[fuse]
```

Using git This uses latest, unreleased development code from git. While we try not to break master, there are no guarantees on anything.

```
# get borg from github
git clone https://github.com/borgbackup/borg.git

virtualenv --python=python3 borg-env
source borg-env/bin/activate   # always before using!

# install borg + dependencies into virtualenv
pip install sphinx  # optional, to build the docs
cd borg
pip install -r requirements.d/development.txt
pip install -r requirements.d/fuse.txt  # optional, for FUSE support
pip install -e .  # in-place editable mode

# optional: run all the tests, on all supported Python versions
# requires fakeroot, available through your package manager
fakeroot -u tox
```

Note: As a developer or power user, you always want to use a virtual environment.

Quick Start

This chapter will get you started with Borg. The first section presents a simple step by step example that uses Borg to backup data. The next section continues by showing how backups can be automated.

Important note about free space

Before you start creating backups, please make sure that there is *always* a good amount of free space on the filesystem that has your backup repository (and also on ~/.cache). A few GB should suffice for most hard-drive sized repositories.

See also *Indexes / Caches memory usage*.

If Borg runs out of disk space, it tries to free as much space as it can while aborting the current operation safely, which allows to free more space by deleting/pruning archives. This mechanism is not bullet-proof though. If you *really* run out of disk space, it can be hard or impossible to free space, because Borg needs free space to operate - even to delete backup archives. There is a `--save-space` option for some commands, but even with that Borg will need free space to operate.

You can use some monitoring process or just include the free space information in your backup log files (you check them regularly anyway, right?).

Also helpful:

- create a big file as a "space reserve", that you can delete to free space
- if you use LVM: use a LV + a filesystem that you can resize later and have some unallocated PEs you can add to the LV.
- consider using quotas
- use *prune* regularly

A step by step example

1. Before a backup can be made a repository has to be initialized:

```
$ borg init /path/to/repo
```

2. Backup the `~/src` and `~/Documents` directories into an archive called *Monday*:

```
$ borg create /path/to/repo::Monday ~/src ~/Documents
```

3. The next day create a new archive called *Tuesday*:

```
$ borg create -v --stats /path/to/repo::Tuesday ~/src ~/Documents
```

   This backup will be a lot quicker and a lot smaller since only new never before seen data is stored. The `--stats` option causes Borg to output statistics about the newly created archive such as the amount of unique data (not shared with other archives):

```
------------------------------------------------------------------------------
Archive name: Tuesday
Archive fingerprint: bd31004d58f51ea06ff735d2e5ac49376901b21d58035f8fb05dbf866566e3c2
Time (start): Tue, 2016-02-16 18:15:11
Time (end):   Tue, 2016-02-16 18:15:11

Duration: 0.19 seconds
Number of files: 127
------------------------------------------------------------------------------
                       Original size      Compressed size    Deduplicated size
This archive:                4.16 MB              4.17 MB             26.78 kB
All archives:                8.33 MB              8.34 MB              4.19 MB

                       Unique chunks         Total chunks
Chunk index:                     132                  261
------------------------------------------------------------------------------
```

4. List all archives in the repository:

```
$ borg list /path/to/repo
Monday                               Mon, 2016-02-15 19:14:44
Tuesday                              Tue, 2016-02-16 19:15:11
```

5. List the contents of the *Monday* archive:

```
$ borg list /path/to/repo::Monday
drwxr-xr-x user   group          0 Mon, 2016-02-15 18:22:30 home/user/Documents
-rw-r--r-- user   group       7961 Mon, 2016-02-15 18:22:30 home/user/Documents/Important.doc
...
```

6. Restore the *Monday* archive:

```
$ borg extract /path/to/repo::Monday
```

7. Recover disk space by manually deleting the *Monday* archive:

```
$ borg delete /path/to/repo::Monday
```

Note: Borg is quiet by default (it works on WARNING log level). Add the `-v` (or `--verbose` or `--info`) option to adjust the log level to INFO and also use options like `--progress` or `--list` to get progress reporting during command execution.

Automating backups

The following example script backs up `/home` and `/var/www` to a remote server. The script also uses the *borg prune* subcommand to maintain a certain number of old archives:

```
#!/bin/sh
REPOSITORY=username@remoteserver.com:backup

# Backup all of /home and /var/www except a few
# excluded directories
borg create -v --stats                            \
    $REPOSITORY::'{hostname}-{now:%Y-%m-%d}'    \
    /home                                         \
    /var/www                                      \
    --exclude '/home/*/.cache'                    \
    --exclude /home/Ben/Music/Justin\ Bieber    \
    --exclude '*.pyc'

# Use the `prune` subcommand to maintain 7 daily, 4 weekly and 6 monthly
# archives of THIS machine. The '{hostname}-' prefix is very important to
# limit prune's operation to this machine's archives and not apply to
# other machine's archives also.
borg prune -v $REPOSITORY --prefix '{hostname}-' \
    --keep-daily=7 --keep-weekly=4 --keep-monthly=6
```

Pitfalls with shell variables and environment variables

This applies to all environment variables you want borg to see, not just `BORG_PASSPHRASE`. The short explanation is: always `export` your variable, and use single quotes if you're unsure of the details of your shell's expansion behavior. E.g.:

```
export BORG_PASSPHRASE='complicated & long'
```

This is because `export` exposes variables to subprocesses, which borg may be one of. More on `export` can be found in the "ENVIRONMENT" section of the bash(1) man page.

Beware of how `sudo` interacts with environment variables. For example, you may be surprised that the following `export` has no effect on your command:

```
export BORG_PASSPHRASE='complicated & long'
sudo ./yourborgwrapper.sh  # still prompts for password
```

For more information, see sudo(8) man page. Hint: see `env_keep` in sudoers(5), or try `sudo BORG_PASSPHRASE='yourphrase' borg` syntax.

Tip: To debug what your borg process is actually seeing, find its PID (`ps aux|grep borg`) and then look into `/proc/<PID>/environ`.

Backup compression

Default is no compression, but we support different methods with high speed or high compression:

If you have a fast repo storage and you want some compression:

```
$ borg create --compression lz4 /path/to/repo::arch ~
```

If you have a less fast repo storage and you want a bit more compression (N=0..9, 0 means no compression, 9 means high compression):

```
$ borg create --compression zlib,N /path/to/repo::arch ~
```

If you have a very slow repo storage and you want high compression (N=0..9, 0 means low compression, 9 means high compression):

```
$ borg create --compression lzma,N /path/to/repo::arch ~
```

You'll need to experiment a bit to find the best compression for your use case. Keep an eye on CPU load and throughput.

Repository encryption

Repository encryption can be enabled or disabled at repository creation time (the default is enabled, with *repokey* method):

```
$ borg init --encryption=none|repokey|keyfile PATH
```

When repository encryption is enabled all data is encrypted using 256-bit AES encryption and the integrity and authenticity is verified using HMAC-SHA256.

All data is encrypted on the client before being written to the repository. This means that an attacker who manages to compromise the host containing an encrypted archive will not be able to access any of the data, even while the backup is being made.

Borg supports different methods to store the AES and HMAC keys.

`repokey` mode The key is stored inside the repository (in its "config" file). Use this mode if you trust in your good passphrase giving you enough protection. The repository server never sees the plaintext key.

keyfile mode The key is stored on your local disk (in ~/.config/borg/keys/). Use this mode if you want "passphrase and having-the-key" security.

In both modes, the key is stored in encrypted form and can be only decrypted by providing the correct passphrase.

For automated backups the passphrase can be specified using the *BORG_PASSPHRASE* environment variable.

Note: Be careful about how you set that environment, see *this note about password environments* for more information.

Warning: The repository data is totally inaccessible without the key and the key passphrase.
Make a backup copy of the key file (keyfile mode) or repo config file (repokey mode) and keep it at a safe place, so you still have the key in case it gets corrupted or lost. Also keep your passphrase at a safe place.
The backup that is encrypted with that key/passphrase won't help you with that, of course.

Remote repositories

Borg can initialize and access repositories on remote hosts if the host is accessible using SSH. This is fastest and easiest when Borg is installed on the remote host, in which case the following syntax is used:

```
$ borg init user@hostname:/path/to/repo
```

or:

```
$ borg init ssh://user@hostname:port//path/to/repo
```

Remote operations over SSH can be automated with SSH keys. You can restrict the use of the SSH keypair by prepending a forced command to the SSH public key in the remote server's *authorized_keys* file. This example will start Borg in server mode and limit it to a specific filesystem path:

```
command="borg serve --restrict-to-path /path/to/repo",no-pty,no-agent-forwarding,no-port-forwarding,
```

If it is not possible to install Borg on the remote host, it is still possible to use the remote host to store a repository by mounting the remote filesystem, for example, using sshfs:

```
$ sshfs user@hostname:/path/to /path/to
$ borg init /path/to/repo
$ fusermount -u /path/to
```

Usage

Borg consists of a number of commands. Each command accepts a number of arguments and options. The following sections will describe each command in detail.

General

Type of log output The log level of the builtin logging configuration defaults to WARNING. This is because we want Borg to be mostly silent and only output warnings, errors and critical messages.

Log levels: DEBUG < INFO < WARNING < ERROR < CRITICAL

Use --debug to set DEBUG log level - to get debug, info, warning, error and critical level output.

Use --info (or -v or --verbose) to set INFO log level - to get info, warning, error and critical level output.

Use `--warning` (default) to set WARNING log level - to get warning, error and critical level output.

Use `--error` to set ERROR log level - to get error and critical level output.

Use `--critical` to set CRITICAL log level - to get critical level output.

While you can set misc. log levels, do not expect that every command will give different output on different log levels - it's just a possibility.

Warning: Options –critical and –error are provided for completeness, their usage is not recommended as you might miss important information.

Warning: While some options (like `--stats` or `--list`) will emit more informational messages, you have to use INFO (or lower) log level to make them show up in log output. Use `-v` or a logging configuration.

Return codes Borg can exit with the following return codes (rc):

```
0 = success (logged as INFO)
1 = warning (operation reached its normal end, but there were warnings -
    you should check the log, logged as WARNING)
2 = error (like a fatal error, a local or remote exception, the operation
    did not reach its normal end, logged as ERROR)
128+N = killed by signal N (e.g. 137 == kill -9)
```

If you use `--show-rc`, the return code is also logged at the indicated level as the last log entry.

Environment Variables Borg uses some environment variables for automation:

General:

BORG_REPO When set, use the value to give the default repository location. If a command needs an archive parameter, you can abbreviate as *::archive*. If a command needs a repository parameter, you can either leave it away or abbreviate as *::*, if a positional parameter is required.

BORG_PASSPHRASE When set, use the value to answer the passphrase question for encrypted repositories.

BORG_DISPLAY_PASSPHRASE When set, use the value to answer the "display the passphrase for verification" question when defining a new passphrase for encrypted repositories.

BORG_LOGGING_CONF When set, use the given filename as INI-style logging configuration.

BORG_RSH When set, use this command instead of `ssh`. This can be used to specify ssh options, such as a custom identity file `ssh -i /path/to/private/key`. See `man ssh` for other options.

BORG_REMOTE_PATH When set, use the given path/filename as remote path (default is "borg"). Using `--remote-path PATH` commandline option overrides the environment variable.

BORG_FILES_CACHE_TTL When set to a numeric value, this determines the maximum "time to live" for the files cache entries (default: 20). The files cache is used to quickly determine whether a file is unchanged.

TMPDIR where temporary files are stored (might need a lot of temporary space for some operations)

Some automatic "answerers" (if set, they automatically answer confirmation questions):

BORG_UNKNOWN_UNENCRYPTED_REPO_ACCESS_IS_OK=no (or =yes) For "Warning: Attempting to access a previously unknown unencrypted repository"

BORG_RELOCATED_REPO_ACCESS_IS_OK=no (or =yes) For "Warning: The repository at location ... was previously located at ..."

BORG_CHECK_I_KNOW_WHAT_I_AM_DOING=NO (or =YES) For "Warning: 'check –repair' is an experimental feature that might result in data loss."

BORG_DELETE_I_KNOW_WHAT_I_AM_DOING=NO (or =YES) For "You requested to completely DELETE the repository *including* all archives it contains:"

Note: answers are case sensitive. setting an invalid answer value might either give the default answer or ask you interactively, depending on whether retries are allowed (they by default are allowed). So please test your scripts interactively before making them a non-interactive script.

Directories:

BORG_KEYS_DIR Default to '~/.config/borg/keys'. This directory contains keys for encrypted repositories.

BORG_CACHE_DIR Default to '~/.cache/borg'. This directory contains the local cache and might need a lot of space for dealing with big repositories).

Building:

BORG_OPENSSL_PREFIX Adds given OpenSSL header file directory to the default locations (setup.py).

BORG_LZ4_PREFIX Adds given LZ4 header file directory to the default locations (setup.py).

Please note:

- be very careful when using the "yes" sayers, the warnings with prompt exist for your / your data's security/safety
- also be very careful when putting your passphrase into a script, make sure it has appropriate file permissions (e.g. mode 600, root:root).

Resource Usage Borg might use a lot of resources depending on the size of the data set it is dealing with.

CPU: It won't go beyond 100% of 1 core as the code is currently single-threaded. Especially higher zlib and lzma compression levels use significant amounts of CPU cycles.

Memory (RAM): The chunks index and the files index are read into memory for performance reasons. Compression, esp. lzma compression with high levels might need substantial amounts of memory.

Temporary files: Reading data and metadata from a FUSE mounted repository will consume about the same space as the deduplicated chunks used to represent them in the repository.

Cache files: Contains the chunks index and files index (plus a compressed collection of single-archive chunk indexes).

Chunks index: Proportional to the amount of data chunks in your repo. Lots of chunks in your repo imply a big chunks index. It is possible to tweak the chunker params (see create options).

Files index: Proportional to the amount of files in your last backup. Can be switched off (see create options), but next backup will be much slower if you do.

Network: If your repository is remote, all deduplicated (and optionally compressed/ encrypted) data of course has to go over the connection (ssh: repo url). If you use a locally mounted network filesystem, additionally some copy operations used for transaction support also go over the connection. If you backup multiple sources to one target repository, additional traffic happens for cache resynchronization.

In case you are interested in more details, please read the internals documentation.

Units To display quantities, Borg takes care of respecting the usual conventions of scale. Disk sizes are displayed in decimal, using powers of ten (so `kB` means 1000 bytes). For memory usage, binary prefixes are used, and are indicated using the IEC binary prefixes, using powers of two (so `KiB` means 1024 bytes).

Date and Time We format date and time conforming to ISO-8601, that is: YYYY-MM-DD and HH:MM:SS (24h clock).

For more information about that, see: https://xkcd.com/1179/

Unless otherwise noted, we display local date and time. Internally, we store and process date and time as UTC.

borg init

```
usage: borg init [-h] [--critical] [--error] [--warning] [--info] [--debug]
                 [--lock-wait N] [--show-rc] [--no-files-cache] [--umask M]
                 [--remote-path PATH] [-e {none,keyfile,repokey}] [-a]
                 [REPOSITORY]

Initialize an empty repository

positional arguments:
  REPOSITORY            repository to create

optional arguments:
  -h, --help            show this help message and exit
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
  --info, -v, --verbose
                        work on log level INFO
  --debug               work on log level DEBUG
  --lock-wait N         wait for the lock, but max. N seconds (default: 1).
  --show-rc             show/log the return code (rc)
  --no-files-cache      do not load/update the file metadata cache used to
                        detect unchanged files
  --umask M             set umask to M (local and remote, default: 0077)
  --remote-path PATH    set remote path to executable (default: "borg")
  -e {none,keyfile,repokey}, --encryption {none,keyfile,repokey}
                        select encryption key mode (default: "repokey")
  -a, --append-only     create an append-only mode repository
```

Description This command initializes an empty repository. A repository is a filesystem directory containing the deduplicated data from zero or more archives. Encryption can be enabled at repository init time.

Examples

```
# Local repository (default is to use encryption in repokey mode)
$ borg init /path/to/repo

# Local repository (no encryption)
$ borg init --encryption=none /path/to/repo

# Remote repository (accesses a remote borg via ssh)
$ borg init user@hostname:backup

# Remote repository (store the key your home dir)
$ borg init --encryption=keyfile user@hostname:backup
```

Important notes about encryption:

It is not recommended to disable encryption. Repository encryption protects you e.g. against the case that an attacker has access to your backup repository.

But be careful with the key / the passphrase:

If you want "passphrase-only" security, use the `repokey` mode. The key will be stored inside the repository (in its "config" file). In above mentioned attack scenario, the attacker will have the key (but not the passphrase).

If you want "passphrase and having-the-key" security, use the `keyfile` mode. The key will be stored in your home directory (in `.config/borg/keys`). In the attack scenario, the attacker who has just access to your repo won't have the key (and also not the passphrase).

Make a backup copy of the key file (`keyfile` mode) or repo config file (`repokey` mode) and keep it at a safe place, so you still have the key in case it gets corrupted or lost. Also keep the passphrase at a safe place. The backup that is encrypted with that key won't help you with that, of course.

Make sure you use a good passphrase. Not too short, not too simple. The real encryption / decryption key is encrypted with / locked by your passphrase. If an attacker gets your key, he can't unlock and use it without knowing the passphrase.

Be careful with special or non-ascii characters in your passphrase:

- Borg processes the passphrase as unicode (and encodes it as utf-8), so it does not have problems dealing with even the strangest characters.
- BUT: that does not necessarily apply to your OS / VM / keyboard configuration.

So better use a long passphrase made from simple ascii chars than one that includes non-ascii stuff or characters that are hard/impossible to enter on a different keyboard layout.

You can change your passphrase for existing repos at any time, it won't affect the encryption/decryption key or other secrets.

borg create

```
usage: borg create [-h] [--critical] [--error] [--warning] [--info] [--debug]
                   [--lock-wait N] [--show-rc] [--no-files-cache] [--umask M]
                   [--remote-path PATH] [-s] [-p] [--list]
                   [--filter STATUSCHARS] [-e PATTERN]
                   [--exclude-from EXCLUDEFILE] [--exclude-caches]
                   [--exclude-if-present FILENAME] [--keep-tag-files]
                   [-c SECONDS] [-x] [--numeric-owner]
                   [--timestamp yyyy-mm-ddThh:mm:ss]
                   [--chunker-params CHUNK_MIN_EXP,CHUNK_MAX_EXP,HASH_MASK_BITS,HASH_WINDOW_SIZE]
                   [--ignore-inode] [-C COMPRESSION] [--read-special] [-n]
                   ARCHIVE PATH [PATH ...]

Create new archive

positional arguments:
  ARCHIVE               name of archive to create (must be also a valid
                        directory name)
  PATH                  paths to archive

optional arguments:
  -h, --help            show this help message and exit
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
  --info, -v, --verbose
```

```
                      work on log level INFO
--debug               work on log level DEBUG
--lock-wait N         wait for the lock, but max. N seconds (default: 1).
--show-rc             show/log the return code (rc)
--no-files-cache      do not load/update the file metadata cache used to
                      detect unchanged files
--umask M             set umask to M (local and remote, default: 0077)
--remote-path PATH    set remote path to executable (default: "borg")
-s, --stats           print statistics for the created archive
-p, --progress        show progress display while creating the archive,
                      showing Original, Compressed and Deduplicated sizes,
                      followed by the Number of files seen and the path
                      being processed, default: False
--list                output verbose list of items (files, dirs, ...)
--filter STATUSCHARS  only display items with the given status characters
-e PATTERN, --exclude PATTERN
                      exclude paths matching PATTERN
--exclude-from EXCLUDEFILE
                      read exclude patterns from EXCLUDEFILE, one per line
--exclude-caches      exclude directories that contain a CACHEDIR.TAG file
                      (http://www.brynosaurus.com/cachedir/spec.html)
--exclude-if-present FILENAME
                      exclude directories that contain the specified file
--keep-tag-files      keep tag files of excluded caches/directories
-c SECONDS, --checkpoint-interval SECONDS
                      write checkpoint every SECONDS seconds (Default: 300)
-x, --one-file-system
                      stay in same file system, do not cross mount points
--numeric-owner       only store numeric user and group identifiers
--timestamp yyyy-mm-ddThh:mm:ss
                      manually specify the archive creation date/time (UTC).
                      alternatively, give a reference file/directory.
--chunker-params CHUNK_MIN_EXP,CHUNK_MAX_EXP,HASH_MASK_BITS,HASH_WINDOW_SIZE
                      specify the chunker parameters. default: 19,23,21,4095
--ignore-inode        ignore inode data in the file metadata cache used to
                      detect unchanged files.
-C COMPRESSION, --compression COMPRESSION
                      select compression algorithm (and level): none == no
                      compression (default), lz4 == lz4, zlib == zlib
                      (default level 6), zlib,0 .. zlib,9 == zlib (with
                      level 0..9), lzma == lzma (default level 6), lzma,0 ..
                      lzma,9 == lzma (with level 0..9).
--read-special        open and read block and char device files as well as
                      FIFOs as if they were regular files. Also follows
                      symlinks pointing to these kinds of files.
-n, --dry-run         do not create a backup archive
```

Description This command creates a backup archive containing all files found while recursively traversing all paths specified. The archive will consume almost no disk space for files or parts of files that have already been stored in other archives.

The archive name needs to be unique. It must not end in '.checkpoint' or '.checkpoint.N' (with N being a number), because these names are used for checkpoints and treated in special ways.

In the archive name, you may use the following format tags: {now}, {utcnow}, {fqdn}, {hostname}, {user}, {pid}, {borgversion}

To speed up pulling backups over sshfs and similar network file systems which do not provide correct inode infor-

mation the –ignore-inode flag can be used. This potentially decreases reliability of change detection, while avoiding always reading all files on these file systems.

See the output of the "borg help patterns" command for more help on exclude patterns. See the output of the "borg help placeholders" command for more help on placeholders.

Examples

```
# Backup ~/Documents into an archive named "my-documents"
$ borg create /path/to/repo::my-documents ~/Documents

# same, but verbosely list all files as we process them
$ borg create -v --list /path/to/repo::my-documents ~/Documents

# Backup ~/Documents and ~/src but exclude pyc files
$ borg create /path/to/repo::my-files \
    ~/Documents                       \
    ~/src                             \
    --exclude '*.pyc'

# Backup home directories excluding image thumbnails (i.e. only
# /home/*/.thumbnails is excluded, not /home/*/*/.thumbnails)
$ borg create /path/to/repo::my-files /home \
    --exclude 're:^/home/[^/]+/\.thumbnails/'

# Do the same using a shell-style pattern
$ borg create /path/to/repo::my-files /home \
    --exclude 'sh:/home/*/.thumbnails'

# Backup the root filesystem into an archive named "root-YYYY-MM-DD"
# use zlib compression (good, but slow) - default is no compression
$ borg create -C zlib,6 /path/to/repo::root-{now:%Y-%m-%d} / --one-file-system

# Make a big effort in fine granular deduplication (big chunk management
# overhead, needs a lot of RAM and disk space, see formula in internals
# docs - same parameters as borg < 1.0 or attic):
$ borg create --chunker-params 10,23,16,4095 /path/to/repo::small /smallstuff

# Backup a raw device (must not be active/in use/mounted at that time)
$ dd if=/dev/sdx bs=10M | borg create /path/to/repo::my-sdx -

# No compression (default)
$ borg create /path/to/repo::arch ~

# Super fast, low compression
$ borg create --compression lz4 /path/to/repo::arch ~

# Less fast, higher compression (N = 0..9)
$ borg create --compression zlib,N /path/to/repo::arch ~

# Even slower, even higher compression (N = 0..9)
$ borg create --compression lzma,N /path/to/repo::arch ~

# Use short hostname, user name and current time in archive name
$ borg create /path/to/repo::{hostname}-{user}-{now} ~
$ borg create /path/to/repo::{hostname}-{user}-{now:%Y-%m-%d_%H:%M:%S} ~
```

borg extract

```
usage: borg extract [-h] [--critical] [--error] [--warning] [--info] [--debug]
                    [--lock-wait N] [--show-rc] [--no-files-cache] [--umask M]
                    [--remote-path PATH] [--list] [-n] [-e PATTERN]
                    [--exclude-from EXCLUDEFILE] [--numeric-owner]
                    [--strip-components NUMBER] [--stdout] [--sparse]
                    ARCHIVE [PATH [PATH ...]]

Extract archive contents

positional arguments:
  ARCHIVE               archive to extract
  PATH                  paths to extract; patterns are supported

optional arguments:
  -h, --help            show this help message and exit
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
  --info, -v, --verbose
                        work on log level INFO
  --debug               work on log level DEBUG
  --lock-wait N         wait for the lock, but max. N seconds (default: 1).
  --show-rc             show/log the return code (rc)
  --no-files-cache      do not load/update the file metadata cache used to
                        detect unchanged files
  --umask M             set umask to M (local and remote, default: 0077)
  --remote-path PATH    set remote path to executable (default: "borg")
  --list                output verbose list of items (files, dirs, ...)
  -n, --dry-run         do not actually change any files
  -e PATTERN, --exclude PATTERN
                        exclude paths matching PATTERN
  --exclude-from EXCLUDEFILE
                        read exclude patterns from EXCLUDEFILE, one per line
  --numeric-owner       only obey numeric user and group identifiers
  --strip-components NUMBER
                        Remove the specified number of leading path elements.
                        Pathnames with fewer elements will be silently
                        skipped.
  --stdout              write all extracted data to stdout
  --sparse              create holes in output sparse file from all-zero
                        chunks
```

Description This command extracts the contents of an archive. By default the entire archive is extracted but a subset of files and directories can be selected by passing a list of `PATHs` as arguments. The file selection can further be restricted by using the `--exclude` option.

See the output of the "borg help patterns" command for more help on exclude patterns.

Examples

```
# Extract entire archive
$ borg extract /path/to/repo::my-files

# Extract entire archive and list files while processing
$ borg extract -v --list /path/to/repo::my-files
```

```
# Extract the "src" directory
$ borg extract /path/to/repo::my-files home/USERNAME/src

# Extract the "src" directory but exclude object files
$ borg extract /path/to/repo::my-files home/USERNAME/src --exclude '*.o'

# Restore a raw device (must not be active/in use/mounted at that time)
$ borg extract --stdout /path/to/repo::my-sdx | dd of=/dev/sdx bs=10M
```

Note: currently, extract always writes into the current working directory (“.”), so make sure you `cd` to the right place before calling `borg extract`.

borg check

```
usage: borg check [-h] [--critical] [--error] [--warning] [--info] [--debug]
                  [--lock-wait N] [--show-rc] [--no-files-cache] [--umask M]
                  [--remote-path PATH] [--repository-only] [--archives-only]
                  [--repair] [--save-space] [--last N] [-P PREFIX]
                  [REPOSITORY_OR_ARCHIVE]

Check repository consistency

positional arguments:
  REPOSITORY_OR_ARCHIVE
                        repository or archive to check consistency of

optional arguments:
  -h, --help            show this help message and exit
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
  --info, -v, --verbose
                        work on log level INFO
  --debug               work on log level DEBUG
  --lock-wait N         wait for the lock, but max. N seconds (default: 1).
  --show-rc             show/log the return code (rc)
  --no-files-cache      do not load/update the file metadata cache used to
                        detect unchanged files
  --umask M             set umask to M (local and remote, default: 0077)
  --remote-path PATH    set remote path to executable (default: "borg")
  --repository-only     only perform repository checks
  --archives-only       only perform archives checks
  --repair              attempt to repair any inconsistencies found
  --save-space          work slower, but using less space
  --last N              only check last N archives (Default: all)
  -P PREFIX, --prefix PREFIX
                        only consider archive names starting with this prefix
```

Description The check command verifies the consistency of a repository and the corresponding archives.

First, the underlying repository data files are checked:

- For all segments the segment magic (header) is checked
- For all objects stored in the segments, all metadata (e.g. crc and size) and all data is read. The read data is checked by size and CRC. Bit rot and other types of accidental damage can be detected this way.

- If we are in repair mode and a integrity error is detected for a segment, we try to recover as many objects from the segment as possible.
- In repair mode, it makes sure that the index is consistent with the data stored in the segments.
- If you use a remote repo server via ssh:, the repo check is executed on the repo server without causing significant network traffic.
- The repository check can be skipped using the –archives-only option.

Second, the consistency and correctness of the archive metadata is verified:

- Is the repo manifest present? If not, it is rebuilt from archive metadata chunks (this requires reading and decrypting of all metadata and data).
- Check if archive metadata chunk is present. if not, remove archive from manifest.
- For all files (items) in the archive, for all chunks referenced by these files, check if chunk is present. If a chunk is not present and we are in repair mode, replace it with a same-size replacement chunk of zeros. If a previously lost chunk reappears (e.g. via a later backup) and we are in repair mode, the all-zero replacement chunk will be replaced by the correct chunk. This requires reading of archive and file metadata, but not data.
- If we are in repair mode and we checked all the archives: delete orphaned chunks from the repo.
- if you use a remote repo server via ssh:, the archive check is executed on the client machine (because if encryption is enabled, the checks will require decryption and this is always done client-side, because key access will be required).
- The archive checks can be time consuming, they can be skipped using the –repository-only option.

borg rename

```
usage: borg rename [-h] [--critical] [--error] [--warning] [--info] [--debug]
                   [--lock-wait N] [--show-rc] [--no-files-cache] [--umask M]
                   [--remote-path PATH]
                   ARCHIVE NEWNAME

Rename an existing archive

positional arguments:
  ARCHIVE               archive to rename
  NEWNAME               the new archive name to use

optional arguments:
  -h, --help            show this help message and exit
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
  --info, -v, --verbose
                        work on log level INFO
  --debug               work on log level DEBUG
  --lock-wait N         wait for the lock, but max. N seconds (default: 1).
  --show-rc             show/log the return code (rc)
  --no-files-cache      do not load/update the file metadata cache used to
                        detect unchanged files
  --umask M             set umask to M (local and remote, default: 0077)
  --remote-path PATH    set remote path to executable (default: "borg")
```

Description This command renames an archive in the repository.

Examples

```
$ borg create /path/to/repo::archivename ~
$ borg list /path/to/repo
archivename                          Mon, 2016-02-15 19:50:19

$ borg rename /path/to/repo::archivename newname
$ borg list /path/to/repo
newname                              Mon, 2016-02-15 19:50:19
```

borg list

```
usage: borg list [-h] [--critical] [--error] [--warning] [--info] [--debug]
                 [--lock-wait N] [--show-rc] [--no-files-cache] [--umask M]
                 [--remote-path PATH] [--short] [--list-format LISTFORMAT]
                 [-P PREFIX]
                 [REPOSITORY_OR_ARCHIVE]

List archive or repository contents

positional arguments:
  REPOSITORY_OR_ARCHIVE
                        repository/archive to list contents of

optional arguments:
  -h, --help            show this help message and exit
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
  --info, -v, --verbose
                        work on log level INFO
  --debug               work on log level DEBUG
  --lock-wait N         wait for the lock, but max. N seconds (default: 1).
  --show-rc             show/log the return code (rc)
  --no-files-cache      do not load/update the file metadata cache used to
                        detect unchanged files
  --umask M             set umask to M (local and remote, default: 0077)
  --remote-path PATH    set remote path to executable (default: "borg")
  --short               only print file/directory names, nothing else
  --list-format LISTFORMAT
                        specify format for archive file listing (default:
                        "{mode} {user:6} {group:6} {size:8d} {isomtime}
                        {path}{extra}{NEWLINE}") Special "{formatkeys}" exists
                        to list available keys
  -P PREFIX, --prefix PREFIX
                        only consider archive names starting with this prefix
```

Description This command lists the contents of a repository or an archive.

Examples

```
$ borg list /path/to/repo
Monday                               Mon, 2016-02-15 19:15:11
repo                                 Mon, 2016-02-15 19:26:54
root-2016-02-15                      Mon, 2016-02-15 19:36:29
newname                              Mon, 2016-02-15 19:50:19
```

```
...

$ borg list /path/to/repo::root-2016-02-15
drwxr-xr-x root   root          0 Mon, 2016-02-15 17:44:27 .
drwxrwxr-x root   root          0 Mon, 2016-02-15 19:04:49 bin
-rwxr-xr-x root   root    1029624 Thu, 2014-11-13 00:08:51 bin/bash
lrwxrwxrwx root   root          0 Fri, 2015-03-27 20:24:26 bin/bzcmp -> bzdiff
-rwxr-xr-x root   root       2140 Fri, 2015-03-27 20:24:22 bin/bzdiff
...

$ borg list /path/to/repo::archiveA --list-format="{mode} {user:6} {group:6} {size:8d} {isomtime} {pa
drwxrwxr-x user   user          0 Sun, 2015-02-01 11:00:00 .
drwxrwxr-x user   user          0 Sun, 2015-02-01 11:00:00 code
drwxrwxr-x user   user          0 Sun, 2015-02-01 11:00:00 code/myproject
-rw-rw-r-- user   user    1416192 Sun, 2015-02-01 11:00:00 code/myproject/file.ext
...

# see what is changed between archives, based on file modification time, size and file path
$ borg list /path/to/repo::archiveA --list-format="{mtime:%s}{TAB}{size}{TAB}{path}{LF}" |sort -n > /
$ borg list /path/to/repo::archiveB --list-format="{mtime:%s}{TAB}{size}{TAB}{path}{LF}" |sort -n > /
$ diff -y /tmp/list.archiveA /tmp/list.archiveB
1422781200      0       .                                       1422781200      0       .
1422781200      0       code                                    1422781200      0       code
1422781200      0       code/myproject                          1422781200      0       code/myproje
1422781200      1416192 code/myproject/file.ext               | 1454664653      1416192 code/myproje
...
```

borg delete

```
usage: borg delete [-h] [--critical] [--error] [--warning] [--info] [--debug]
                   [--lock-wait N] [--show-rc] [--no-files-cache] [--umask M]
                   [--remote-path PATH] [-p] [-s] [-c] [--force]
                   [--save-space]
                   [TARGET]

Delete an existing repository or archive

positional arguments:
  TARGET                archive or repository to delete

optional arguments:
  -h, --help            show this help message and exit
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
  --info, -v, --verbose
                        work on log level INFO
  --debug               work on log level DEBUG
  --lock-wait N         wait for the lock, but max. N seconds (default: 1).
  --show-rc             show/log the return code (rc)
  --no-files-cache      do not load/update the file metadata cache used to
                        detect unchanged files
  --umask M             set umask to M (local and remote, default: 0077)
  --remote-path PATH    set remote path to executable (default: "borg")
  -p, --progress        show progress display while deleting a single archive
  -s, --stats           print statistics for the deleted archive
  -c, --cache-only      delete only the local cache for the given repository
```

```
  --force               force deletion of corrupted archives
  --save-space          work slower, but using less space
```

Description This command deletes an archive from the repository or the complete repository. Disk space is reclaimed accordingly. If you delete the complete repository, the local cache for it (if any) is also deleted.

Examples

```
# delete a single backup archive:
$ borg delete /path/to/repo::Monday

# delete the whole repository and the related local cache:
$ borg delete /path/to/repo
You requested to completely DELETE the repository *including* all archives it contains:
repo                                 Mon, 2016-02-15 19:26:54
root-2016-02-15                      Mon, 2016-02-15 19:36:29
newname                              Mon, 2016-02-15 19:50:19
Type 'YES' if you understand this and want to continue: YES
```

borg prune

```
usage: borg prune [-h] [--critical] [--error] [--warning] [--info] [--debug]
                  [--lock-wait N] [--show-rc] [--no-files-cache] [--umask M]
                  [--remote-path PATH] [-n] [--force] [-s] [--list]
                  [--keep-within WITHIN] [-H HOURLY] [-d DAILY] [-w WEEKLY]
                  [-m MONTHLY] [-y YEARLY] [-P PREFIX] [--save-space]
                  [REPOSITORY]

Prune repository archives according to specified rules

positional arguments:
  REPOSITORY            repository to prune

optional arguments:
  -h, --help            show this help message and exit
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
  --info, -v, --verbose
                        work on log level INFO
  --debug               work on log level DEBUG
  --lock-wait N         wait for the lock, but max. N seconds (default: 1).
  --show-rc             show/log the return code (rc)
  --no-files-cache      do not load/update the file metadata cache used to
                        detect unchanged files
  --umask M             set umask to M (local and remote, default: 0077)
  --remote-path PATH    set remote path to executable (default: "borg")
  -n, --dry-run         do not change repository
  --force               force pruning of corrupted archives
  -s, --stats           print statistics for the deleted archive
  --list                output verbose list of archives it keeps/prunes
  --keep-within WITHIN  keep all archives within this time interval
  -H HOURLY, --keep-hourly HOURLY
                        number of hourly archives to keep
  -d DAILY, --keep-daily DAILY
```

```
                      number of daily archives to keep
  -w WEEKLY, --keep-weekly WEEKLY
                      number of weekly archives to keep
  -m MONTHLY, --keep-monthly MONTHLY
                      number of monthly archives to keep
  -y YEARLY, --keep-yearly YEARLY
                      number of yearly archives to keep
  -P PREFIX, --prefix PREFIX
                      only consider archive names starting with this prefix
  --save-space        work slower, but using less space
```

Description The prune command prunes a repository by deleting all archives not matching any of the specified retention options. This command is normally used by automated backup scripts wanting to keep a certain number of historic backups.

As an example, "-d 7" means to keep the latest backup on each day, up to 7 most recent days with backups (days without backups do not count). The rules are applied from hourly to yearly, and backups selected by previous rules do not count towards those of later rules. The time that each backup starts is used for pruning purposes. Dates and times are interpreted in the local timezone, and weeks go from Monday to Sunday. Specifying a negative number of archives to keep means that there is no limit.

The "–keep-within" option takes an argument of the form "<int><char>", where char is "H", "d", "w", "m", "y". For example, "–keep-within 2d" means to keep all archives that were created within the past 48 hours. "1m" is taken to mean "31d". The archives kept with this option do not count towards the totals specified by any other options.

If a prefix is set with -P, then only archives that start with the prefix are considered for deletion and only those archives count towards the totals specified by the rules. Otherwise, *all* archives in the repository are candidates for deletion!

Examples Be careful, prune is a potentially dangerous command, it will remove backup archives.

The default of prune is to apply to **all archives in the repository** unless you restrict its operation to a subset of the archives using `--prefix`. When using `--prefix`, be careful to choose a good prefix - e.g. do not use a prefix "foo" if you do not also want to match "foobar".

It is strongly recommended to always run `prune -v --list --dry-run ...` first so you will see what it would do without it actually doing anything.

There is also a visualized prune example in `docs/misc/prune-example.txt`.

```
# Keep 7 end of day and 4 additional end of week archives.
# Do a dry-run without actually deleting anything.
$ borg prune -v --list --dry-run --keep-daily=7 --keep-weekly=4 /path/to/repo

# Same as above but only apply to archive names starting with the hostname
# of the machine followed by a "-" character:
$ borg prune -v --list --keep-daily=7 --keep-weekly=4 --prefix='{hostname}-' /path/to/repo

# Keep 7 end of day, 4 additional end of week archives,
# and an end of month archive for every month:
$ borg prune -v --list --keep-daily=7 --keep-weekly=4 --keep-monthly=-1 /path/to/repo

# Keep all backups in the last 10 days, 4 additional end of week archives,
# and an end of month archive for every month:
$ borg prune -v --list --keep-within=10d --keep-weekly=4 --keep-monthly=-1 /path/to/repo
```

borg info

```
usage: borg info [-h] [--critical] [--error] [--warning] [--info] [--debug]
                 [--lock-wait N] [--show-rc] [--no-files-cache] [--umask M]
                 [--remote-path PATH]
                 ARCHIVE

Show archive details such as disk space used

positional arguments:
  ARCHIVE               archive to display information about

optional arguments:
  -h, --help            show this help message and exit
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
  --info, -v, --verbose
                        work on log level INFO
  --debug               work on log level DEBUG
  --lock-wait N         wait for the lock, but max. N seconds (default: 1).
  --show-rc             show/log the return code (rc)
  --no-files-cache      do not load/update the file metadata cache used to
                        detect unchanged files
  --umask M             set umask to M (local and remote, default: 0077)
  --remote-path PATH    set remote path to executable (default: "borg")
```

Description This command displays some detailed information about the specified archive.

Examples

```
$ borg info /path/to/repo::root-2016-02-15
Name: root-2016-02-15
Fingerprint: 57c827621f21b000a8d363c1e163cc55983822b3afff3a96df595077a660be50
Hostname: myhostname
Username: root
Time (start): Mon, 2016-02-15 19:36:29
Time (end):   Mon, 2016-02-15 19:39:26
Command line: /usr/local/bin/borg create -v --list -C zlib,6 /path/to/repo::root-2016-02-15 / --one-
Number of files: 38100

                       Original size      Compressed size    Deduplicated size
This archive:                1.33 GB            613.25 MB            571.64 MB
All archives:                1.63 GB            853.66 MB            584.12 MB

                       Unique chunks         Total chunks
Chunk index:                   36858                48844
```

borg mount

```
usage: borg mount [-h] [--critical] [--error] [--warning] [--info] [--debug]
                  [--lock-wait N] [--show-rc] [--no-files-cache] [--umask M]
                  [--remote-path PATH] [-f] [-o OPTIONS]
                  REPOSITORY_OR_ARCHIVE MOUNTPOINT
```

```
Mount archive or an entire repository as a FUSE fileystem

positional arguments:
  REPOSITORY_OR_ARCHIVE
                        repository/archive to mount
  MOUNTPOINT            where to mount filesystem

optional arguments:
  -h, --help            show this help message and exit
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
  --info, -v, --verbose
                        work on log level INFO
  --debug               work on log level DEBUG
  --lock-wait N         wait for the lock, but max. N seconds (default: 1).
  --show-rc             show/log the return code (rc)
  --no-files-cache      do not load/update the file metadata cache used to
                        detect unchanged files
  --umask M             set umask to M (local and remote, default: 0077)
  --remote-path PATH    set remote path to executable (default: "borg")
  -f, --foreground      stay in foreground, do not daemonize
  -o OPTIONS            Extra mount options
```

Description This command mounts an archive as a FUSE filesystem. This can be useful for browsing an archive or restoring individual files. Unless the `--foreground` option is given the command will run in the background until the filesystem is `umounted`.

The BORG_MOUNT_DATA_CACHE_ENTRIES environment variable is meant for advanced users to tweak the performance. It sets the number of cached data chunks; additional memory usage can be up to ~8 MiB times this number. The default is the number of CPU cores.

For mount options, see the fuse(8) manual page. Additional mount options supported by borg:

- allow_damaged_files: by default damaged files (where missing chunks were replaced with runs of zeros by borg check –repair) are not readable and return EIO (I/O error). Set this option to read such files.

Examples

```
$ borg mount /path/to/repo::root-2016-02-15 /tmp/mymountpoint
$ ls /tmp/mymountpoint
bin  boot  etc      home  lib  lib64  lost+found  media  mnt  opt  root  sbin  srv  tmp  usr  var
$ fusermount -u /tmp/mymountpoint
```

borg change-passphrase

```
usage: borg change-passphrase [-h] [--critical] [--error] [--warning] [--info]
                              [--debug] [--lock-wait N] [--show-rc]
                              [--no-files-cache] [--umask M]
                              [--remote-path PATH]
                              [REPOSITORY]

Change repository key file passphrase

positional arguments:
  REPOSITORY
```

```
optional arguments:
  -h, --help            show this help message and exit
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
  --info, -v, --verbose
                        work on log level INFO
  --debug               work on log level DEBUG
  --lock-wait N         wait for the lock, but max. N seconds (default: 1).
  --show-rc             show/log the return code (rc)
  --no-files-cache      do not load/update the file metadata cache used to
                        detect unchanged files
  --umask M             set umask to M (local and remote, default: 0077)
  --remote-path PATH    set remote path to executable (default: "borg")
```

Description The key files used for repository encryption are optionally passphrase protected. This command can be used to change this passphrase.

Examples

```
# Create a key file protected repository
$ borg init --encryption=keyfile -v /path/to/repo
Initializing repository at "/path/to/repo"
Enter new passphrase:
Enter same passphrase again:
Remember your passphrase. Your data will be inaccessible without it.
Key in "/root/.config/borg/keys/mnt_backup" created.
Keep this key safe. Your data will be inaccessible without it.
Synchronizing chunks cache...
Archives: 0, w/ cached Idx: 0, w/ outdated Idx: 0, w/o cached Idx: 0.
Done.

# Change key file passphrase
$ borg change-passphrase -v /path/to/repo
Enter passphrase for key /root/.config/borg/keys/mnt_backup:
Enter new passphrase:
Enter same passphrase again:
Remember your passphrase. Your data will be inaccessible without it.
Key updated
```

borg serve

```
usage: borg serve [-h] [--critical] [--error] [--warning] [--info] [--debug]
                  [--lock-wait N] [--show-rc] [--no-files-cache] [--umask M]
                  [--remote-path PATH] [--restrict-to-path PATH]
                  [--append-only]

Start in server mode. This command is usually not used manually.


optional arguments:
  -h, --help            show this help message and exit
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
```

```
  --info, -v, --verbose
                        work on log level INFO
  --debug               work on log level DEBUG
  --lock-wait N         wait for the lock, but max. N seconds (default: 1).
  --show-rc             show/log the return code (rc)
  --no-files-cache      do not load/update the file metadata cache used to
                        detect unchanged files
  --umask M             set umask to M (local and remote, default: 0077)
  --remote-path PATH    set remote path to executable (default: "borg")
  --restrict-to-path PATH
                        restrict repository access to PATH
  --append-only         only allow appending to repository segment files
```

Description This command starts a repository server process. This command is usually not used manually.

Examples borg serve has special support for ssh forced commands (see `authorized_keys` example below): it will detect that you use such a forced command and extract the value of the `--restrict-to-path` option(s). It will then parse the original command that came from the client, makes sure that it is also `borg serve` and enforce path restriction(s) as given by the forced command. That way, other options given by the client (like `--info` or `--umask`) are preserved (and are not fixed by the forced command).

```
# Allow an SSH keypair to only run borg, and only have access to /path/to/repo.
# Use key options to disable unneeded and potentially dangerous SSH functionality.
# This will help to secure an automated remote backup system.
$ cat ~/.ssh/authorized_keys
command="borg serve --restrict-to-path /path/to/repo",no-pty,no-agent-forwarding,no-port-forwarding,r
```

borg upgrade

```
usage: borg upgrade [-h] [--critical] [--error] [--warning] [--info] [--debug]
                    [--lock-wait N] [--show-rc] [--no-files-cache] [--umask M]
                    [--remote-path PATH] [-p] [-n] [-i]
                    [REPOSITORY]

upgrade a repository from a previous version

positional arguments:
  REPOSITORY            path to the repository to be upgraded

optional arguments:
  -h, --help            show this help message and exit
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
  --info, -v, --verbose
                        work on log level INFO
  --debug               work on log level DEBUG
  --lock-wait N         wait for the lock, but max. N seconds (default: 1).
  --show-rc             show/log the return code (rc)
  --no-files-cache      do not load/update the file metadata cache used to
                        detect unchanged files
  --umask M             set umask to M (local and remote, default: 0077)
  --remote-path PATH    set remote path to executable (default: "borg")
  -p, --progress        show progress display while upgrading the repository
```

```
  -n, --dry-run         do not change repository
  -i, --inplace         rewrite repository in place, with no chance of going
                        back to older versions of the repository.
```

Description Upgrade an existing Borg repository. This currently supports converting an Attic repository to Borg and also helps with converting Borg 0.xx to 1.0.

Currently, only LOCAL repositories can be upgraded (issue #465).

It will change the magic strings in the repository's segments to match the new Borg magic strings. The keyfiles found in $ATTIC_KEYS_DIR or ~/.attic/keys/ will also be converted and copied to $BORG_KEYS_DIR or ~/.config/borg/keys.

The cache files are converted, from $ATTIC_CACHE_DIR or ~/.cache/attic to $BORG_CACHE_DIR or ~/.cache/borg, but the cache layout between Borg and Attic changed, so it is possible the first backup after the conversion takes longer than expected due to the cache resync.

Upgrade should be able to resume if interrupted, although it will still iterate over all segments. If you want to start from scratch, use *borg delete* over the copied repository to make sure the cache files are also removed:

    borg delete borg

Unless `--inplace` is specified, the upgrade process first creates a backup copy of the repository, in REPOSITORY.upgrade-DATETIME, using hardlinks. This takes longer than in place upgrades, but is much safer and gives progress information (as opposed to `cp -al`). Once you are satisfied with the conversion, you can safely destroy the backup copy.

WARNING: Running the upgrade in place will make the current copy unusable with older version, with no way of going back to previous versions. This can PERMANENTLY DAMAGE YOUR REPOSITORY! Attic CAN NOT READ BORG REPOSITORIES, as the magic strings have changed. You have been warned.

Examples

```
# Upgrade the borg repository to the most recent version.
$ borg upgrade -v /path/to/repo
making a hardlink copy in /path/to/repo.upgrade-2016-02-15-20:51:55
opening attic repository with borg and converting
no key file found for repository
converting repo index /path/to/repo/index.0
converting 1 segments...
converting borg 0.xx to borg current
no key file found for repository
```

borg break-lock

```
usage: borg break-lock [-h] [--critical] [--error] [--warning] [--info]
                       [--debug] [--lock-wait N] [--show-rc]
                       [--no-files-cache] [--umask M] [--remote-path PATH]
                       [REPOSITORY]

Break the repository lock (e.g. in case it was left by a dead borg.

positional arguments:
  REPOSITORY            repository for which to break the locks

optional arguments:
  -h, --help            show this help message and exit
```

```
  --critical            work on log level CRITICAL
  --error               work on log level ERROR
  --warning             work on log level WARNING (default)
  --info, -v, --verbose
                        work on log level INFO
  --debug               work on log level DEBUG
  --lock-wait N         wait for the lock, but max. N seconds (default: 1).
  --show-rc             show/log the return code (rc)
  --no-files-cache      do not load/update the file metadata cache used to
                        detect unchanged files
  --umask M             set umask to M (local and remote, default: 0077)
  --remote-path PATH    set remote path to executable (default: "borg")
```

Description This command breaks the repository and cache locks. Please use carefully and only while no borg process (on any machine) is trying to access the Cache or the Repository.

Miscellaneous Help

borg help patterns Exclusion patterns support four separate styles, fnmatch, shell, regular expressions and path prefixes. By default, fnmatch is used. If followed by a colon (':') the first two characters of a pattern are used as a style selector. Explicit style selection is necessary when a non-default style is desired or when the desired pattern starts with two alphanumeric characters followed by a colon (i.e. *aa:something/**).

Fnmatch, selector *fm:*

> This is the default style. These patterns use a variant of shell pattern syntax, with '*' matching any number of characters, '?' matching any single character, '[...]' matching any single character specified, including ranges, and '[!...]' matching any character not specified. For the purpose of these patterns, the path separator ('' for Windows and '/' on other systems) is not treated specially. Wrap meta-characters in brackets for a literal match (i.e. *[?]* to match the literal character *?*). For a path to match a pattern, it must completely match from start to end, or must match from the start to just before a path separator. Except for the root path, paths will never end in the path separator when matching is attempted. Thus, if a given pattern ends in a path separator, a '*' is appended before matching is attempted.

Shell-style patterns, selector *sh:*

> Like fnmatch patterns these are similar to shell patterns. The difference is that the pattern may include **/ for matching zero or more directory levels, * for matching zero or more arbitrary characters with the exception of any path separator.

Regular expressions, selector *re:*

> Regular expressions similar to those found in Perl are supported. Unlike shell patterns regular expressions are not required to match the complete path and any substring match is sufficient. It is strongly recommended to anchor patterns to the start ('^'), to the end ('$') or both. Path separators ('' for Windows and '/' on other systems) in paths are always normalized to a forward slash ('/') before applying a pattern. The regular expression syntax is described in the Python documentation for the re module.

Prefix path, selector *pp:*

> This pattern style is useful to match whole sub-directories. The pattern *pp:/data/bar* matches */data/bar* and everything therein.

Exclusions can be passed via the command line option *–exclude*. When used from within a shell the patterns should be quoted to protect them from expansion.

The *–exclude-from* option permits loading exclusion patterns from a text file with one pattern per line. Lines empty or starting with the number sign ('#') after removing whitespace on both ends are ignored. The optional style selector

prefix is also supported for patterns loaded from a file. Due to whitespace removal paths with whitespace at the beginning or end can only be excluded using regular expressions.

Examples:

```
# Exclude '/home/user/file.o' but not '/home/user/file.odt':
$ borg create -e '*.o' backup /

# Exclude '/home/user/junk' and '/home/user/subdir/junk' but
# not '/home/user/importantjunk' or '/etc/junk':
$ borg create -e '/home/*/junk' backup /

# Exclude the contents of '/home/user/cache' but not the directory itself:
$ borg create -e /home/user/cache/ backup /

# The file '/home/user/cache/important' is *not* backed up:
$ borg create -e /home/user/cache/ backup / /home/user/cache/important

# The contents of directories in '/home' are not backed up when their name
# ends in '.tmp'
$ borg create --exclude 're:^/home/[^/]+\.tmp/' backup /

# Load exclusions from file
$ cat >exclude.txt <<EOF
# Comment line
/home/*/junk
*.tmp
fm:aa:something/*
re:^/home/[^/]\.tmp/
sh:/home/*/.thumbnails
EOF
$ borg create --exclude-from exclude.txt backup /
```

borg help placeholders

Repository (or Archive) URLs, –prefix and –remote-path values support these placeholders:

{hostname}

The (short) hostname of the machine.

{fqdn}

The full name of the machine.

{now}

The current local date and time.

{utcnow}

The current UTC date and time.

{user}

The user name (or UID, if no name is available) of the user running borg.

{pid}

The current process ID.

{borgversion}

The version of borg.

Examples:

```
borg create /path/to/repo::{hostname}-{user}-{utcnow} ...
borg create /path/to/repo::{hostname}-{now:%Y-%m-%d_%H:%M:%S} ...
borg prune --prefix '{hostname}-' ...
```

Debug Commands

There are some more commands (all starting with "debug-") which are all **not intended for normal use** and **potentially very dangerous** if used incorrectly.

They exist to improve debugging capabilities without direct system access, e.g. in case you ever run into some severe malfunction. Use them only if you know what you are doing or if a trusted Borg developer tells you what to do.

Additional Notes

Here are misc. notes about topics that are maybe not covered in enough detail in the usage section.

Item flags `borg create -v --list` outputs a verbose list of all files, directories and other file system items it considered (no matter whether they had content changes or not). For each item, it prefixes a single-letter flag that indicates type and/or status of the item.

If you are interested only in a subset of that output, you can give e.g. `--filter=AME` and it will only show regular files with A, M or E status (see below).

A uppercase character represents the status of a regular file relative to the "files" cache (not relative to the repo – this is an issue if the files cache is not used). Metadata is stored in any case and for 'A' and 'M' also new data chunks are stored. For 'U' all data chunks refer to already existing chunks.

- 'A' = regular file, added (see also *I am seeing 'A' (added) status for a unchanged file!?* in the FAQ)
- 'M' = regular file, modified
- 'U' = regular file, unchanged
- 'E' = regular file, an error happened while accessing/reading *this* file

A lowercase character means a file type other than a regular file, borg usually just stores their metadata:

- 'd' = directory
- 'b' = block device
- 'c' = char device
- 'h' = regular file, hardlink (to already seen inodes)
- 's' = symlink
- 'f' = fifo

Other flags used include:

- 'i' = backup data was read from standard input (stdin)
- '-' = dry run, item was *not* backed up
- '?' = missing status code (if you see this, please file a bug report!)

–chunker-params The chunker params influence how input files are cut into pieces (chunks) which are then considered for deduplication. They also have a big impact on resource usage (RAM and disk space) as the amount of resources needed is (also) determined by the total amount of chunks in the repository (see *Indexes / Caches memory usage* for details).

`--chunker-params=10,23,16,4095` results in a fine-grained deduplication and creates a big amount of chunks and thus uses a lot of resources to manage them. This is good for relatively small data volumes and if the machine has a good amount of free RAM and disk space.

`--chunker-params=19,23,21,4095` (default) results in a coarse-grained deduplication and creates a much smaller amount of chunks and thus uses less resources. This is good for relatively big data volumes and if the machine has a relatively low amount of free RAM and disk space.

If you already have made some archives in a repository and you then change chunker params, this of course impacts deduplication as the chunks will be cut differently.

In the worst case (all files are big and were touched in between backups), this will store all content into the repository again.

Usually, it is not that bad though:

- usually most files are not touched, so it will just re-use the old chunks it already has in the repo
- files smaller than the (both old and new) minimum chunksize result in only one chunk anyway, so the resulting chunks are same and deduplication will apply

If you switch chunker params to save resources for an existing repo that already has some backup archives, you will see an increasing effect over time, when more and more files have been touched and stored again using the bigger chunksize **and** all references to the smaller older chunks have been removed (by deleting / pruning archives).

If you want to see an immediate big effect on resource usage, you better start a new repository when changing chunker params.

For more details, see *Chunks*.

–read-special The –read-special option is special - you do not want to use it for normal full-filesystem backups, but rather after carefully picking some targets for it.

The option `--read-special` triggers special treatment for block and char device files as well as FIFOs. Instead of storing them as such a device (or FIFO), they will get opened, their content will be read and in the backup archive they will show up like a regular file.

Symlinks will also get special treatment if (and only if) they point to such a special file: instead of storing them as a symlink, the target special file will get processed as described above.

One intended use case of this is backing up the contents of one or multiple block devices, like e.g. LVM snapshots or inactive LVs or disk partitions.

You need to be careful about what you include when using `--read-special`, e.g. if you include `/dev/zero`, your backup will never terminate.

Restoring such files' content is currently only supported one at a time via `--stdout` option (and you have to redirect stdout to where ever it shall go, maybe directly into an existing device file of your choice or indirectly via `dd`).

To some extent, mounting a backup archive with the backups of special files via `borg mount` and then loop-mounting the image files from inside the mount point will work. If you plan to access a lot of data in there, it likely will scale and perform better if you do not work via the FUSE mount.

Example Imagine you have made some snapshots of logical volumes (LVs) you want to backup.

Note: For some scenarios, this is a good method to get "crash-like" consistency (I call it crash-like because it is the same as you would get if you just hit the reset button or your machine would abrubtly and completely crash). This is better than no consistency at all and a good method for some use cases, but likely not good enough if you have databases running.

Then you create a backup archive of all these snapshots. The backup process will see a "frozen" state of the logical volumes, while the processes working in the original volumes continue changing the data stored there.

You also add the output of `lvdisplay` to your backup, so you can see the LV sizes in case you ever need to recreate and restore them.

After the backup has completed, you remove the snapshots again.

```
$ # create snapshots here
$ lvdisplay > lvdisplay.txt
$ borg create --read-special /path/to/repo::arch lvdisplay.txt /dev/vg0/*-snapshot
$ # remove snapshots here
```

Now, let's see how to restore some LVs from such a backup.

```
$ borg extract /path/to/repo::arch lvdisplay.txt
$ # create empty LVs with correct sizes here (look into lvdisplay.txt).
$ # we assume that you created an empty root and home LV and overwrite it now:
$ borg extract --stdout /path/to/repo::arch dev/vg0/root-snapshot > /dev/vg0/root
$ borg extract --stdout /path/to/repo::arch dev/vg0/home-snapshot > /dev/vg0/home
```

Append-only mode A repository can be made "append-only", which means that Borg will never overwrite or delete committed data. This is useful for scenarios where multiple machines back up to a central backup server using `borg serve`, since a hacked machine cannot delete backups permanently.

To activate append-only mode, edit the repository `config` file and add a line `append_only=1` to the `[repository]` section (or edit the line if it exists).

In append-only mode Borg will create a transaction log in the `transactions` file, where each line is a transaction and a UTC timestamp.

In addition, `borg serve` can act as if a repository is in append-only mode with its option `--append-only`. This can be very useful for fine-tuning access control in `.ssh/authorized_keys`

```
command="borg serve --append-only ..." ssh-rsa <key used for not-always-trustable backup clients>
command="borg serve ..." ssh-rsa <key used for backup management>
```

Example Suppose an attacker remotely deleted all backups, but your repository was in append-only mode. A transaction log in this situation might look like this:

```
transaction 1, UTC time 2016-03-31T15:53:27.383532
transaction 5, UTC time 2016-03-31T15:53:52.588922
transaction 11, UTC time 2016-03-31T15:54:23.887256
transaction 12, UTC time 2016-03-31T15:55:54.022540
transaction 13, UTC time 2016-03-31T15:55:55.472564
```

From your security logs you conclude the attacker gained access at 15:54:00 and all the backups where deleted or replaced by compromised backups. From the log you know that transactions 11 and later are compromised. Note that the transaction ID is the name of the *last* file in the transaction. For example, transaction 11 spans files 6 to 11.

In a real attack you'll likely want to keep the compromised repository intact to analyze what the attacker tried to achieve. It's also a good idea to make this copy just in case something goes wrong during the recovery. Since recovery is done by deleting some files, a hard link copy (`cp -al`) is sufficient.

The first step to reset the repository to transaction 5, the last uncompromised transaction, is to remove the `hints.N` and `index.N` files in the repository (these two files are always expendable). In this example N is 13.

Then remove or move all segment files from the segment directories in `data/` starting with file 6:

```
rm data/**/{6..13}
```

That's all to it.

Drawbacks As data is only appended, and nothing deleted, commands like `prune` or `delete` won't free disk space, they merely tag data as deleted in a new transaction.

Note that you can go back-and-forth between normal and append-only operation by editing the configuration file, it's not a "one way trip".

Further considerations Append-only mode is not respected by tools other than Borg. `rm` still works on the repository. Make sure that backup client machines only get to access the repository via `borg serve`.

Ensure that no remote access is possible if the repository is temporarily set to normal mode for e.g. regular pruning.

Further protections can be implemented, but are outside of Borgs scope. For example, file system snapshots or wrapping `borg serve` to set special permissions or ACLs on new data files.

Deployment

This chapter will give an example how to setup a borg repository server for multiple clients.

Machines

There are multiple machines used in this chapter and will further be named by their respective fully qualified domain name (fqdn).

- The backup server: *backup01.srv.local*
- The clients:
  - John Doe's desktop: *johndoe.clnt.local*
  - Webserver 01: *web01.srv.local*
  - Application server 01: *app01.srv.local*

User and group

The repository server needs to have only one UNIX user for all the clients. Recommended user and group with additional settings:

- User: *backup*
- Group: *backup*
- Shell: */bin/bash* (or other capable to run the *borg serve* command)
- Home: */home/backup*

Most clients shall initiate a backup from the root user to catch all users, groups and permissions (e.g. when backing up */home*).

Folders

The following folder tree layout is suggested on the repository server:

- User home directory, /home/backup
- Repositories path (storage pool): /home/backup/repos
- Clients restricted paths (*/home/backup/repos/<client fqdn>*):
  - johndoe.clnt.local: */home/backup/repos/johndoe.clnt.local*
  - web01.srv.local: */home/backup/repos/web01.srv.local*
  - app01.srv.local: */home/backup/repos/app01.srv.local*

Restrictions

Borg is instructed to restrict clients into their own paths: `borg serve --restrict-to-path /home/backup/repos/<client fqdn>`

There is only one ssh key per client allowed. Keys are added for `johndoe.clnt.local`, `web01.srv.local` and `app01.srv.local`. But they will access the backup under only one UNIX user account as: `backup@backup01.srv.local`. Every key in `$HOME/.ssh/authorized_keys` has a forced command and restrictions applied as shown below:

```
command="cd /home/backup/repos/<client fqdn>;
         borg serve --restrict-to-path /home/backup/repos/<client fqdn>",
         no-port-forwarding,no-X11-forwarding,no-pty,
         no-agent-forwarding,no-user-rc <keytype> <key> <host>
```

Note: The text shown above needs to be written on a single line!

The options which are added to the key will perform the following:

1. Change working directory
2. Run `borg serve` restricted to the client base path
3. Restrict ssh and do not allow stuff which imposes a security risk

Due to the `cd` command we use, the server automatically changes the current working directory. Then client doesn't need to have knowledge of the absolute or relative remote repository path and can directly access the repositories at `<user>@<host>:<repo>`.

Note: The setup above ignores all client given commandline parameters which are normally appended to the *borg serve* command.

Client

The client needs to initialize the *pictures* repository like this:

borg init backup@backup01.srv.local:pictures

Or with the full path (should actually never be used, as only for demonstrational purposes). The server should automatically change the current working directory to the *<client fqdn>* folder.

borg init backup@backup01.srv.local:/home/backup/repos/johndoe.clnt.local/pictures

When *johndoe.clnt.local* tries to access a not restricted path the following error is raised. John Doe tries to backup into the Web 01 path:

borg init backup@backup01.srv.local:/home/backup/repos/web01.srv.local/pictures

```
~~~ SNIP ~~~
Remote: borg.remote.PathNotAllowed: /home/backup/repos/web01.srv.local/pictures
~~~ SNIP ~~~
Repository path not allowed
```

Ansible

Ansible takes care of all the system-specific commands to add the user, create the folder. Even when the configuration is changed the repository server configuration is satisfied and reproducible.

Automate setting up an repository server with the user, group, folders and permissions a Ansible playbook could be used. Keep in mind the playbook uses the Arch Linux pacman package manager to install and keep borg up-to-date.

```
- hosts: backup01.srv.local
  vars:
    user: backup
    group: backup
    home: /home/backup
    pool: "{{ home }}/repos"
    auth_users:
      - host: johndoe.clnt.local
        key: "{{ lookup('file', '/path/to/keys/johndoe.clnt.local.pub') }}"
      - host: web01.clnt.local
        key: "{{ lookup('file', '/path/to/keys/web01.clnt.local.pub') }}"
      - host: app01.clnt.local
        key: "{{ lookup('file', '/path/to/keys/app01.clnt.local.pub') }}"
  tasks:
  - pacman: name=borg state=latest update_cache=yes
  - group: name="{{ group }}" state=present
  - user: name="{{ user }}" shell=/bin/bash home="{{ home }}" createhome=yes group="{{ group }}" grou
  - file: path="{{ home }}" owner="{{ user }}" group="{{ group }}" mode=0700 state=directory
  - file: path="{{ home }}/.ssh" owner="{{ user }}" group="{{ group }}" mode=0700 state=directory
  - file: path="{{ pool }}" owner="{{ user }}" group="{{ group }}" mode=0700 state=directory
  - authorized_key: user="{{ user }}"
                    key="{{ item.key }}"
                    key_options='command="cd {{ pool }}/{{ item.host }};borg serve --restrict-to-path
    with_items: auth_users
  - file: path="{{ home }}/.ssh/authorized_keys" owner="{{ user }}" group="{{ group }}" mode=0600 sta
  - file: path="{{ pool }}/{{ item.host }}" owner="{{ user }}" group="{{ group }}" mode=0700 state=di
    with_items: auth_users
```

Enhancements

As this chapter only describes a simple and effective setup it could be further enhanced when supporting (a limited set) of client supplied commands. A wrapper for starting *borg serve* could be written. Or borg itself could be enhanced to

autodetect it runs under SSH by checking the *SSH_ORIGINAL_COMMAND* environment variable. This is left open for future improvements.

When extending ssh autodetection in borg no external wrapper script is necessary and no other interpreter or application has to be deployed.

See also

- SSH Daemon manpage
- Ansible

Frequently asked questions

Can I backup VM disk images?

Yes, the deduplication technique used by Borg makes sure only the modified parts of the file are stored. Also, we have optional simple sparse file support for extract.

Can I backup from multiple servers into a single repository?

Yes, but in order for the deduplication used by Borg to work, it needs to keep a local cache containing checksums of all file chunks already stored in the repository. This cache is stored in `~/.cache/borg/`. If Borg detects that a repository has been modified since the local cache was updated it will need to rebuild the cache. This rebuild can be quite time consuming.

So, yes it's possible. But it will be most efficient if a single repository is only modified from one place. Also keep in mind that Borg will keep an exclusive lock on the repository while creating or deleting archives, which may make *simultaneous* backups fail.

Can I copy or synchronize my repo to another location?

Yes, you could just copy all the files. Make sure you do that while no backup is running. So what you get here is this:

- client machine —borg create—> repo1
- repo1 —copy—> repo2

There is no special borg command to do the copying, just use cp or rsync if you want to do that.

But think about whether that is really what you want. If something goes wrong in repo1, you will have the same issue in repo2 after the copy.

If you want to have 2 independent backups, it is better to do it like this:

- client machine —borg create—> repo1
- client machine —borg create—> repo2

Which file types, attributes, etc. are *not* preserved?

- UNIX domain sockets (because it does not make sense - they are meaningless without the running process that created them and the process needs to recreate them in any case). So, don't panic if your backup misses a UDS!

- The precise on-disk (or rather: not-on-disk) representation of the holes in a sparse file. Archive creation has no special support for sparse files, holes are backed up as (deduplicated and compressed) runs of zero bytes. Archive extraction has optional support to extract all-zero chunks as holes in a sparse file.
- filesystem specific attributes, like ext4 immutable bit, see #618.

Are there other known limitations?

- A single archive can only reference a limited volume of file/dir metadata, usually corresponding to tens or hundreds of millions of files/dirs. When trying to go beyond that limit, you will get a fatal IntegrityError exception telling that the (archive) object is too big. An easy workaround is to create multiple archives with less items each. See also the *Note about archive limitations* and #1452.

Why is my backup bigger than with attic? Why doesn't Borg do compression by default?

Attic was rather unflexible when it comes to compression, it always compressed using zlib level 6 (no way to switch compression off or adjust the level or algorithm).

Borg offers a lot of different compression algorithms and levels. Which of them is the best for you pretty much depends on your use case, your data, your hardware – so you need to do an informed decision about whether you want to use compression, which algorithm and which level you want to use. This is why compression defaults to none.

How can I specify the encryption passphrase programmatically?

The encryption passphrase can be specified programmatically using the *BORG_PASSPHRASE* environment variable. This is convenient when setting up automated encrypted backups. Another option is to use key file based encryption with a blank passphrase. See *Repository encryption* for more details.

Note: Be careful how you set the environment; using the `env` command, a `system()` call or using inline shell scripts might expose the credentials in the process list directly and they will be readable to all users on a system. Using `export` in a shell script file should be safe, however, as the environment of a process is accessible only to that user.

When backing up to remote encrypted repos, is encryption done locally?

Yes, file and directory metadata and data is locally encrypted, before leaving the local machine. We do not mean the transport layer encryption by that, but the data/metadata itself. Transport layer encryption (e.g. when ssh is used as a transport) applies additionally.

When backing up to remote servers, do I have to trust the remote server?

Yes and No.

No, as far as data confidentiality is concerned - if you use encryption, all your files/dirs data and metadata are stored in their encrypted form into the repository.

Yes, as an attacker with access to the remote server could delete (or otherwise make unavailable) all your backups.

How can I protect against a hacked backup client?

Assume you backup your backup client machine C to the backup server S and C gets hacked. In a simple push setup, the attacker could then use borg on C to delete all backups residing on S.

These are your options to protect against that:

- Do not allow to permanently delete data from the repo, see *Append-only mode*.
- Use a pull-mode setup using `ssh -R`, see #900.
- Mount C's filesystem on another machine and then create a backup of it.
- Do not give C filesystem-level access to S.

How can I protect against a hacked backup server?

Just in case you got the impression that pull-mode backups are way more safe than push-mode, you also need to consider the case that your backup server S gets hacked. In case S has access to a lot of clients C, that might bring you into even bigger trouble than a hacked backup client in the previous FAQ entry.

These are your options to protect against that:

- Use the standard push-mode setup (see also previous FAQ entry).
- Mount (the repo part of) S's filesystem on C.
- Do not give S file-system level access to C.
- Have your backup server at a well protected place (maybe not reachable from the internet), configure it safely, apply security updates, monitor it, ...

How can I protect against theft, sabotage, lightning, fire, ...?

In general: if your only backup medium is nearby the backupped machine and always connected, you can easily get into trouble: they likely share the same fate if something goes really wrong.

Thus:

- have multiple backup media
- have media disconnected from network, power, computer
- have media at another place
- have a relatively recent backup on your media

Why do I get "connection closed by remote" after a while?

When doing a backup to a remote server (using a ssh: repo URL), it sometimes stops after a while (some minutes, hours, ... - not immediately) with "connection closed by remote" error message. Why?

That's a good question and we are trying to find a good answer in #636.

Why am I seeing idle borg serve processes on the repo server?

Maybe the ssh connection between client and server broke down and that was not yet noticed on the server. Try these settings:

```
# /etc/ssh/sshd_config on borg repo server - kill connection to client
# after ClientAliveCountMax * ClientAliveInterval seconds with no response
ClientAliveInterval 20
ClientAliveCountMax 3
```

If you have multiple borg create ... ; borg create ... commands in a already serialized way in a single script, you need to give them –lock-wait N (with N being a bit more than the time the server needs to terminate broken down connections and release the lock).

The borg cache eats way too much disk space, what can I do?

There is a temporary (but maybe long lived) hack to avoid using lots of disk space for chunks.archive.d (see #235 for details):

```
# this assumes you are working with the same user as the backup.
# you can get the REPOID from the "config" file inside the repository.
cd ~/.cache/borg/<REPOID>
rm -rf chunks.archive.d ; touch chunks.archive.d
```

This deletes all the cached archive chunk indexes and replaces the directory that kept them with a file, so borg won't be able to store anything "in" there in future.

This has some pros and cons, though:

- much less disk space needs for ~/.cache/borg.
- chunk cache resyncs will be slower as it will have to transfer chunk usage metadata for all archives from the repository (which might be slow if your repo connection is slow) and it will also have to build the hashtables from that data. chunk cache resyncs happen e.g. if your repo was written to by another machine (if you share same backup repo between multiple machines) or if your local chunks cache was lost somehow.

The long term plan to improve this is called "borgception", see #474.

If a backup stops mid-way, does the already-backed-up data stay there?

Yes, Borg supports resuming backups.

During a backup a special checkpoint archive named `<archive-name>.checkpoint` is saved every checkpoint interval (the default value for this is 5 minutes) containing all the data backed-up until that point.

Checkpoints only happen between files (so they don't help for interruptions happening while a very large file is being processed).

This checkpoint archive is a valid archive (all files in it are valid and complete), but it is only a partial backup (not all files that you wanted to backup are contained in it). Having it in the repo until a successful, full backup is completed is useful because it references all the transmitted chunks up to the checkpoint. This means that in case of an interruption, you only need to retransfer the data since the last checkpoint.

If a backup was interrupted, you do not need to do any special considerations, just invoke `borg create` as you always do. You may use the same archive name as in previous attempt or a different one (e.g. if you always include the current datetime), it does not matter.

Borg always does full single-pass backups, so it will start again from the beginning - but it will be much faster, because some of the data was already stored into the repo (and is still referenced by the checkpoint archive), so it does not need to get transmitted and stored again.

Once your backup has finished successfully, you can delete all `<archive-name>.checkpoint` archives.

How can I backup huge file(s) over a unstable connection?

You can use this "split trick" as a workaround for the in-between-files-only checkpoints (see above), huge files and a instable connection to the repository:

Split the huge file(s) into parts of manageable size (e.g. 100MB) and create a temporary archive of them. Borg will create checkpoints now more frequently than if you try to backup the files in their original form (e.g. 100GB).

After that, you can remove the parts again and backup the huge file(s) in their original form. This will now work a lot faster as a lot of content chunks are already in the repository.

After you have successfully backed up the huge original file(s), you can remove the temporary archive you made from the parts.

We realize that this is just a better-than-nothing workaround, see #1198 for a potential solution.

Please note that this workaround only helps you for backup, not for restore.

If it crashes with a UnicodeError, what can I do?

Check if your encoding is set correctly. For most POSIX-like systems, try:

```
export LANG=en_US.UTF-8  # or similar, important is correct charset
```

I can't extract non-ascii filenames by giving them on the commandline!?

This might be due to different ways to represent some characters in unicode or due to other non-ascii encoding issues.

If you run into that, try this:

- avoid the non-ascii characters on the commandline by e.g. extracting the parent directory (or even everything)
- mount the repo using FUSE and use some file manager

Can Borg add redundancy to the backup data to deal with hardware malfunction?

No, it can't. While that at first sounds like a good idea to defend against some defect HDD sectors or SSD flash blocks, dealing with this in a reliable way needs a lot of low-level storage layout information and control which we do not have (and also can't get, even if we wanted).

So, if you need that, consider RAID or a filesystem that offers redundant storage or just make backups to different locations / different hardware.

See also #225.

Can Borg verify data integrity of a backup archive?

Yes, if you want to detect accidental data damage (like bit rot), use the `check` operation. It will notice corruption using CRCs and hashes. If you want to be able to detect malicious tampering also, use an encrypted repo. It will then be able to check using CRCs and HMACs.

I am seeing 'A' (added) status for a unchanged file!?

The files cache is used to determine whether Borg already "knows" / has backed up a file and if so, to skip the file from chunking. It does intentionally *not* contain files that have a modification time (mtime) same as the newest mtime in the created archive.

So, if you see an 'A' status for unchanged file(s), they are likely the files with the most recent mtime in that archive.

This is expected: it is to avoid data loss with files that are backed up from a snapshot and that are immediately changed after the snapshot (but within mtime granularity time, so the mtime would not change). Without the code that removes these files from the files cache, the change that happened right after the snapshot would not be contained in the next backup as Borg would think the file is unchanged.

This does not affect deduplication, the file will be chunked, but as the chunks will often be the same and already stored in the repo (except in the above mentioned rare condition), it will just re-use them as usual and not store new data chunks.

If you want to avoid unnecessary chunking, just create or touch a small or empty file in your backup source file set (so that one has the latest mtime, not your 50GB VM disk image) and, if you do snapshots, do the snapshot after that.

Since only the files cache is used in the display of files status, those files are reported as being added when, really, chunks are already used.

It always chunks all my files, even unchanged ones!

Borg maintains a files cache where it remembers the mtime, size and inode of files. When Borg does a new backup and starts processing a file, it first looks whether the file has changed (compared to the values stored in the files cache). If the values are the same, the file is assumed unchanged and thus its contents won't get chunked (again).

Borg can't keep an infinite history of files of course, thus entries in the files cache have a "maximum time to live" which is set via the environment variable BORG_FILES_CACHE_TTL (and defaults to 20). Every time you do a backup (on the same machine, using the same user), the cache entries' ttl values of files that were not "seen" are incremented by 1 and if they reach BORG_FILES_CACHE_TTL, the entry is removed from the cache.

So, for example, if you do daily backups of 26 different data sets A, B, C, ..., Z on one machine (using the default TTL), the files from A will be already forgotten when you repeat the same backups on the next day and it will be slow because it would chunk all the files each time. If you set BORG_FILES_CACHE_TTL to at least 26 (or maybe even a small multiple of that), it would be much faster.

Another possible reason is that files don't always have the same path, for example if you mount a filesystem without stable mount points for each backup. If the directory where you mount a filesystem is different every time, Borg assume they are different files.

Is there a way to limit bandwidth with Borg?

There is no command line option to limit bandwidth with Borg, but bandwidth limiting can be accomplished with pipeviewer:

Create a wrapper script: /usr/local/bin/pv-wrapper

```
#!/bin/bash
    ## -q, --quiet              do not output any transfer information at all
    ## -L, --rate-limit RATE    limit transfer to RATE bytes per second
export RATE=307200
pv -q -L $RATE  | "$@"
```

Add BORG_RSH environment variable to use pipeviewer wrapper script with ssh.

```
export BORG_RSH='/usr/local/bin/pv-wrapper.sh ssh'
```

Now Borg will be bandwidth limited. Nice thing about pv is that you can change rate-limit on the fly:

```
pv -R $(pidof pv) -L 102400
```

I am having troubles with some network/FUSE/special filesystem, why?

Borg is doing nothing special in the filesystem, it only uses very common and compatible operations (even the locking is just "mkdir").

So, if you are encountering issues like slowness, corruption or malfunction when using a specific filesystem, please try if you can reproduce the issues with a local (non-network) and proven filesystem (like ext4 on Linux).

If you can't reproduce the issue then, you maybe have found an issue within the filesystem code you used (not with Borg). For this case, it is recommended that you talk to the developers / support of the network fs and maybe open an issue in their issue tracker. Do not file an issue in the Borg issue tracker.

If you can reproduce the issue with the proven filesystem, please file an issue in the Borg issue tracker about that.

Requirements for the borg single-file binary, esp. (g)libc?

We try to build the binary on old, but still supported systems - to keep the minimum requirement for the (g)libc low. The (g)libc can't be bundled into the binary as it needs to fit your kernel and OS, but Python and all other required libraries will be bundled into the binary.

If your system fulfills the minimum (g)libc requirement (see the README that is released with the binary), there should be no problem. If you are slightly below the required version, maybe just try. Due to the dynamic loading (or not loading) of some shared libraries, it might still work depending on what libraries are actually loaded and used.

In the borg git repository, there is scripts/glibc_check.py that can determine (based on the symbols' versions they want to link to) whether a set of given (Linux) binaries works with a given glibc version.

Why was Borg forked from Attic?

Borg was created in May 2015 in response to the difficulty of getting new code or larger changes incorporated into Attic and establishing a bigger developer community / more open development.

More details can be found in ticket 217 that led to the fork.

Borg intends to be:

- simple:
  - as simple as possible, but no simpler
  - do the right thing by default, but offer options
- open:

  - welcome feature requests
  - accept pull requests of good quality and coding style
  - give feedback on PRs that can't be accepted "as is"
  - discuss openly, don't work in the dark
- changing:
  - Borg is not compatible with Attic
  - do not break compatibility accidentally, without a good reason or without warning. allow compatibility breaking for other cases.
  - if major version number changes, it may have incompatible changes

Support

Please first read the docs, the existing issue tracker issues and mailing list posts – a lot of stuff is already documented / explained / discussed / filed there.

Issue Tracker

If you've found a bug or have a concrete feature request, please create a new ticket on the project's issue tracker.

For more general questions or discussions, IRC or mailing list are preferred.

Chat (IRC)

Join us on channel #borgbackup on chat.freenode.net.

As usual on IRC, just ask or tell directly and then patiently wait for replies. Stay connected.

You could use the following link (after connecting, you can change the random nickname you get by typing "/nick mydesirednickname"):

http://webchat.freenode.net/?randomnick=1&channels=%23borgbackup&uio=MTY9dHJ1ZSY5PXRydWUa8

Mailing list

To find out about the mailing list, its topic, how to subscribe, how to unsubscribe and where you can find the archives of the list, see the mailing list homepage.

Bounties and Fundraisers

We use BountySource to allow monetary contributions to the project and the developers, who push it forward.

There, you can give general funds to the borgbackup members (the developers will then spend the funds as they deem fit). If you do not have some specific bounty (see below), you can use this as a general way to say "Thank You!" and support the software / project you like.

If you want to encourage developers to fix some specific issue or implement some specific feature suggestion, you can post a new bounty or back an existing one (they always refer to an issue in our issue tracker).

As a developer, you can become a Bounty Hunter and win bounties (earn money) by contributing to Borg, a free and open source software project.

We might also use BountySource to fund raise for some bigger goals.

Resources

This is a collection of additional resources that are somehow related to borgbackup.

Videos, Talks, Presentations

Some of them refer to attic, but you can do the same stuff (and more) with borgbackup.

- BorgBackup Installation and Basic Usage (english screencast)
- TW's slides for borgbackup talks / lightning talks (just grab the latest ones)
- Attic / Borg Backup talk from GPN 2015 (media.ccc.de)
- Attic / Borg Backup talk from GPN 2015 (youtube)
- Attic talk from Easterhegg 2015 (media.ccc.de)
- Attic talk from Easterhegg 2015 (youtube)
- Attic Backup: Mount your encrypted backups over ssh (youtube)
- Evolution of Borg (youtube)

Software

- BorgWeb - a very simple web UI for BorgBackup
- some other stuff found at the BorgBackup Github organisation
- borgmatic - simple wrapper script for BorgBackup that creates and prunes backups

Changelog

Important note about pre-1.0.4 potential repo corruption

Some external errors (like network or disk I/O errors) could lead to corruption of the backup repository due to issue #1138.

A sign that this happened is if "E" status was reported for a file that can not be explained by problems with the source file. If you still have logs from "borg create -v –list", you can check for "E" status.

Here is what could cause corruption and what you can do now:

1. I/O errors (e.g. repo disk errors) while writing data to repo.

This could lead to corrupted segment files.

Fix:

```
# check for corrupt chunks / segments:
borg check -v --repository-only REPO

# repair the repo:
borg check -v --repository-only --repair REPO
```

```
# make sure everything is fixed:
borg check -v --repository-only REPO
```

2. Unreliable network / unreliable connection to the repo.

This could lead to archive metadata corruption.

Fix:

```
# check for corrupt archives:
borg check -v --archives-only REPO

# delete the corrupt archives:
borg delete --force REPO::CORRUPT_ARCHIVE

# make sure everything is fixed:
borg check -v --archives-only REPO
```

3. In case you want to do more intensive checking.

The best check that everything is ok is to run a dry-run extraction:

```
borg extract -v --dry-run REPO::ARCHIVE
```

Version 1.0.7 (2016-08-19)

Security fixes:

- borg serve: fix security issue with remote repository access, #1428 If you used e.g. –restrict-to-path /path/client1/ (with or without trailing slash does not make a difference), it acted like a path prefix match using /path/client1 (note the missing trailing slash) - the code then also allowed working in e.g. /path/client13 or /path/client1000.

  As this could accidentally lead to major security/privacy issues depending on the pathes you use, the behaviour was changed to be a strict directory match. That means –restrict-to-path /path/client1 (with or without trailing slash does not make a difference) now uses /path/client1/ internally (note the trailing slash here!) for matching and allows precisely that path AND any path below it. So, /path/client1 is allowed, /path/client1/repo1 is allowed, but not /path/client13 or /path/client1000.

  If you willingly used the undocumented (dangerous) previous behaviour, you may need to rearrange your –restrict-to-path pathes now. We are sorry if that causes work for you, but we did not want a potentially dangerous behaviour in the software (not even using a for-backwards-compat option).

Bug fixes:

- fixed repeated LockTimeout exceptions when borg serve tried to write into a already write-locked repo (e.g. by a borg mount), #502 part b) This was solved by the fix for #1220 in 1.0.7rc1 already.
- fix cosmetics + file leftover for “not a valid borg repository”, #1490
- Cache: release lock if cache is invalid, #1501
- borg extract –strip-components: fix leak of preloaded chunk contents
- Repository, when a InvalidRepository exception happens:
  - fix spurious, empty lock.roster
  - fix repo not closed cleanly

New features:

- implement borg debug-info, fixes #1122 (just calls already existing code via cli, same output as below tracebacks)

Other changes:

- skip the O_NOATIME test on GNU Hurd, fixes #1315 (this is a very minor issue and the GNU Hurd project knows the bug)
- document using a clean repo to test / build the release

Version 1.0.7rc2 (2016-08-13)

Bug fixes:

- do not write objects to repository that are bigger than the allowed size, borg will reject reading them, #1451.

  Important: if you created archives with many millions of files or directories, please verify if you can open them successfully, e.g. try a "borg list REPO::ARCHIVE".
- lz4 compression: dynamically enlarge the (de)compression buffer, the static buffer was not big enough for archives with extremely many items, #1453
- larger item metadata stream chunks, raise archive item limit by 8x, #1452
- fix untracked segments made by moved DELETEs, #1442

  Impact: Previously (metadata) segments could become untracked when deleting data, these would never be cleaned up.
- extended attributes (xattrs) related fixes:
  - fixed a race condition in xattrs querying that led to the entire file not being backed up (while logging the error, exit code = 1), #1469
  - fixed a race condition in xattrs querying that led to a crash, #1462
  - raise OSError including the error message derived from errno, deal with path being a integer FD

Other changes:

- print active env var override by default, #1467
- xattr module: refactor code, deduplicate, clean up
- repository: split object size check into too small and too big
- add a transaction_id assertion, so borg init on a broken (inconsistent) filesystem does not look like a coding error in borg, but points to the real problem.
- explain confusing TypeError caused by compat support for old servers, #1456
- add forgotten usage help file from build_usage
- refactor/unify buffer code into helpers.Buffer class, add tests
- docs:
  - document archive limitation, #1452
  - improve prune examples

Version 1.0.7rc1 (2016-08-05)

Bug fixes:

- fix repo lock deadlocks (related to lock upgrade), #1220
- catch unpacker exceptions, resync, #1351
- fix borg break-lock ignoring BORG_REPO env var, #1324
- files cache performance fixes (fixes unneccessary re-reading/chunking/ hashing of unmodified files for some use cases):
  - fix unintended file cache eviction, #1430
  - implement BORG_FILES_CACHE_TTL, update FAQ, raise default TTL from 10 to 20, #1338
- FUSE:
  - cache partially read data chunks (performance), #965, #966
  - always create a root dir, #1125
- use an OrderedDict for helptext, making the build reproducible, #1346
- RemoteRepository init: always call close on exceptions, #1370 (cosmetic)
- ignore stdout/stderr broken pipe errors (cosmetic), #1116

New features:

- better borg versions management support (useful esp. for borg servers wanting to offer multiple borg versions and for clients wanting to choose a specific server borg version), #1392:
  - add BORG_VERSION environment variable before executing "borg serve" via ssh
  - add new placeholder {borgversion}
  - substitute placeholders in –remote-path
- borg init –append-only option (makes using the more secure append-only mode more convenient. when used remotely, this requires 1.0.7+ also on the borg server), #1291.

Other changes:

- Vagrantfile:
  - darwin64: upgrade to FUSE for macOS 3.4.1 (aka osxfuse), #1378
  - xenial64: use user "ubuntu", not "vagrant" (as usual), #1331
- tests:
  - fix fuse tests on OS X, #1433
- docs:
  - FAQ: add backup using stable filesystem names recommendation
  - FAQ about glibc compatibility added, #491, glibc-check improved
  - FAQ: 'A' unchanged file; remove ambiguous entry age sentence.
  - OS X: install pkg-config to build with FUSE support, fixes #1400
  - add notes about shell/sudo pitfalls with env. vars, #1380
  - added platform feature matrix
- implement borg debug-dump-repo-objs

Version 1.0.6 (2016-07-12)

Bug fixes:

- Linux: handle multiple LD_PRELOAD entries correctly, #1314, #1111
- Fix crash with unclear message if the libc is not found, #1314, #1111

Other changes:

- tests:
  - Fixed O_NOATIME tests for Solaris and GNU Hurd, #1315
  - Fixed sparse file tests for (file) systems not supporting it, #1310
- docs:
  - Fixed syntax highlighting, #1313
  - misc docs: added data processing overview picture

Version 1.0.6rc1 (2016-07-10)

New features:

- borg check –repair: heal damaged files if missing chunks re-appear (e.g. if the previously missing chunk was added again in a later backup archive), #148. (*) Also improved logging.

Bug fixes:

- sync_dir: silence fsync() failing with EINVAL, #1287 Some network filesystems (like smbfs) don't support this and we use this in repository code.
- borg mount (FUSE):
  - fix directories being shadowed when contained paths were also specified, #1295
  - raise I/O Error (EIO) on damaged files (unless -o allow_damaged_files is used), #1302. (*)
- borg extract: warn if a damaged file is extracted, #1299. (*)
- Added some missing return code checks (ChunkIndex._add, hashindex_resize).
- borg check: fix/optimize initial hash table size, avoids resize of the table.

Other changes:

- tests:
  - add more FUSE tests, #1284
  - deduplicate fuse (u)mount code
  - fix borg binary test issues, #862
- docs:
  - changelog: added release dates to older borg releases
  - fix some sphinx (docs generator) warnings, #881

Notes:

(*) Some features depend on information (chunks_healthy list) added to item metadata when a file with missing chunks was "repaired" using all-zero replacement chunks. The chunks_healthy list is generated since borg 1.0.4, thus borg can't recognize such "repaired" (but content-damaged) files if the repair was done with an older borg version.

Version 1.0.5 (2016-07-07)

Bug fixes:

- borg mount: fix FUSE crash in xattr code on Linux introduced in 1.0.4, #1282

Other changes:

- backport some FAQ entries from master branch
- add release helper scripts
- Vagrantfile:
  - centos6: no FUSE, don't build binary
  - add xz for redhat-like dists

Version 1.0.4 (2016-07-07)

New features:

- borg serve –append-only, #1168 This was included because it was a simple change (append-only functionality was already present via repository config file) and makes better security now practically usable.
- BORG_REMOTE_PATH environment variable, #1258 This was included because it was a simple change (–remote-path cli option was already present) and makes borg much easier to use if you need it.
- Repository: cleanup incomplete transaction on "no space left" condition. In many cases, this can avoid a 100% full repo filesystem (which is very problematic as borg always needs free space - even to delete archives).

Bug fixes:

- Fix wrong handling and reporting of OSErrors in borg create, #1138. This was a serious issue: in the context of "borg create", errors like repository I/O errors (e.g. disk I/O errors, ssh repo connection errors) were handled badly and did not lead to a crash (which would be good for this case, because the repo transaction would be incomplete and trigger a transaction rollback to clean up). Now, error handling for source files is cleanly separated from every other error handling, so only problematic input files are logged and skipped.
- Implement fail-safe error handling for borg extract. Note that this isn't nearly as critical as the borg create error handling bug, since nothing is written to the repo. So this was "merely" misleading error reporting.
- Add missing error handler in directory attr restore loop.
- repo: make sure write data hits disk before the commit tag (#1236) and also sync the containing directory.
- FUSE: getxattr fail must use errno.ENOATTR, #1126 (fixes Mac OS X Finder malfunction: "zero bytes" file length, access denied)
- borg check –repair: do not lose information about the good/original chunks. If we do not lose the original chunk IDs list when "repairing" a file (replacing missing chunks with all-zero chunks), we have a chance to "heal" the file back into its original state later, in case the chunks re-appear (e.g. in a fresh backup). Healing is not implemented yet, see #148.
- fixes for –read-special mode:
  - ignore known files cache, #1241
  - fake regular file mode, #1214
  - improve symlinks handling, #1215
- remove passphrase from subprocess environment, #1105

- Ignore empty index file (will trigger index rebuild), #1195
- add missing placeholder support for –prefix, #1027
- improve exception handling for placeholder replacement
- catch and format exceptions in arg parsing
- helpers: fix "undefined name 'e'" in exception handler
- better error handling for missing repo manifest, #1043
- borg delete:
  - make it possible to delete a repo without manifest
  - borg delete –forced allows to delete corrupted archives, #1139
- borg check:
  - make borg check work for empty repo
  - fix resync and msgpacked item qualifier, #1135
  - rebuild_manifest: fix crash if 'name' or 'time' key were missing.
  - better validation of item metadata dicts, #1130
  - better validation of archive metadata dicts
- close the repo on exit - even if rollback did not work, #1197. This is rather cosmetic, it avoids repo closing in the destructor.
- tests:
  - fix sparse file test, #1170
  - flake8: ignore new F405, #1185
  - catch "invalid argument" on cygwin, #257
  - fix sparseness assertion in test prep, #1264

Other changes:

- make borg build/work on OpenSSL 1.0 and 1.1, #1187
- docs / help:
  - fix / clarify prune help, #1143
  - fix "patterns" help formatting
  - add missing docs / help about placeholders
  - resources: rename atticmatic to borgmatic
  - document sshd settings, #545
  - more details about checkpoints, add split trick, #1171
  - support docs: add freenode web chat link, #1175
  - add prune visualization / example, #723
  - add note that Fnmatch is default, #1247
  - make clear that lzma levels > 6 are a waste of cpu cycles
  - add a "do not edit" note to auto-generated files, #1250

  - update cygwin installation docs
- repository interoperability with borg master (1.1dev) branch:
  - borg check: read item metadata keys from manifest, #1147
  - read v2 hints files, #1235
  - fix hints file "unknown version" error handling bug
- tests: add tests for format_line
- llfuse: update version requirement for freebsd
- Vagrantfile:
  - use openbsd 5.9, #716
  - do not install llfuse on netbsd (broken)
  - update OSXfuse to version 3.3.3
  - use Python 3.5.2 to build the binaries
- glibc compatibility checker: scripts/glibc_check.py
- add .eggs to .gitignore

Version 1.0.3 (2016-05-20)

Bug fixes:

- prune: avoid that checkpoints are kept and completed archives are deleted in a prune run), #997
- prune: fix commandline argument validation - some valid command lines were considered invalid (annoying, but harmless), #942
- fix capabilities extraction on Linux (set xattrs last, after chown()), #1069
- repository: fix commit tags being seen in data
- when probing key files, do binary reads. avoids crash when non-borg binary files are located in borg's key files directory.
- handle SIGTERM and make a clean exit - avoids orphan lock files.
- repository cache: don't cache large objects (avoid using lots of temp. disk space), #1063

Other changes:

- Vagrantfile: OS X: update osxfuse / install lzma package, #933
- setup.py: add check for platform_darwin.c
- setup.py: on freebsd, use a llfuse release that builds ok
- docs / help:
  - update readthedocs URLs, #991
  - add missing docs for "borg break-lock", #992
  - borg create help: add some words to about the archive name
  - borg create help: document format tags, #894

Version 1.0.2 (2016-04-16)

Bug fixes:

- fix malfunction and potential corruption on (nowadays rather rare) big-endian architectures or bi-endian archs in (rare) BE mode. #886, #889

  cache resync / index merge was malfunctioning due to this, potentially leading to data loss. borg info had cosmetic issues (displayed wrong values).

  note: all (widespread) little-endian archs (like x86/x64) or bi-endian archs in (widespread) LE mode (like ARMEL, MIPSEL, ...) were NOT affected.

- add overflow and range checks for 1st (special) uint32 of the hashindex values, switch from int32 to uint32.
- fix so that refcount will never overflow, but just stick to max. value after a overflow would have occured.
- borg delete: fix –cache-only for broken caches, #874

  Makes –cache-only idempotent: it won't fail if the cache is already deleted.

- fixed borg create –one-file-system erroneously traversing into other filesystems (if starting fs device number was 0), #873
- workround a bug in Linux fadvise FADV_DONTNEED, #907

Other changes:

- better test coverage for hashindex, incl. overflow testing, checking correct computations so endianness issues would be discovered.
- reproducible doc for ProgressIndicator*, make the build reproducible.
- use latest llfuse for vagrant machines
- docs:
  - use /path/to/repo in examples, fixes #901
  - fix confusing usage of "repo" as archive name (use "arch")

Version 1.0.1 (2016-04-08)

New features:

Usually there are no new features in a bugfix release, but these were added due to their high impact on security/safety/speed or because they are fixes also:

- append-only mode for repositories, #809, #36 (see docs)
- borg create: add –ignore-inode option to make borg detect unmodified files even if your filesystem does not have stable inode numbers (like sshfs and possibly CIFS).
- add options –warning, –error, –critical for missing log levels, #826. it's not recommended to suppress warnings or errors, but the user may decide this on his own. note: –warning is not given to borg serve so a <= 1.0.0 borg will still work as server (it is not needed as it is the default). do not use –error or –critical when using a <= 1.0.0 borg server.

Bug fixes:

- fix silently skipping EIO, #748
- add context manager for Repository (avoid orphan repository locks), #285
- do not sleep for >60s while waiting for lock, #773

- unpack file stats before passing to FUSE
- fix build on illumos
- don't try to backup doors or event ports (Solaris and derivates)
- remove useless/misleading libc version display, #738
- test suite: reset exit code of persistent archiver, #844
- RemoteRepository: clean up pipe if remote open() fails
- Remote: don't print tracebacks for Error exceptions handled downstream, #792
- if BORG_PASSPHRASE is present but wrong, don't prompt for password, but fail instead, #791
- ArchiveChecker: move "orphaned objects check skipped" to INFO log level, #826
- fix capitalization, add ellipses, change log level to debug for 2 messages, #798

Other changes:

- update llfuse requirement, llfuse 1.0 works
- update OS / dist packages on build machines, #717
- prefer showing –info over -v in usage help, #859
- docs:
  - fix cygwin requirements (gcc-g++)
  - document how to debug / file filesystem issues, #664
  - fix reproducible build of api docs
  - RTD theme: CSS !important overwrite, #727
  - Document logo font. Recreate logo png. Remove GIMP logo file.

Version 1.0.0 (2016-03-05)

The major release number change (0.x -> 1.x) indicates bigger incompatible changes, please read the compatibility notes, adapt / test your scripts and check your backup logs.

Compatibility notes:

- drop support for python 3.2 and 3.3, require 3.4 or 3.5, #221 #65 #490 note: we provide binaries that include python 3.5.1 and everything else needed. they are an option in case you are stuck with < 3.4 otherwise.
- change encryption to be on by default (using "repokey" mode)
- moved keyfile keys from ~/.borg/keys to ~/.config/borg/keys, you can either move them manually or run "borg upgrade <REPO>"
- remove support for –encryption=passphrase, use borg migrate-to-repokey to switch to repokey mode, #97
- remove deprecated –compression <number>, use –compression zlib,<number> instead in case of 0, you could also use –compression none
- remove deprecated –hourly/daily/weekly/monthly/yearly use –keep-hourly/daily/weekly/monthly/yearly instead
- remove deprecated –do-not-cross-mountpoints, use –one-file-system instead
- disambiguate -p option, #563:

  - -p now is same as –progress
  - -P now is same as –prefix
- remove deprecated “borg verify”, use “borg extract –dry-run” instead
- cleanup environment variable semantics, #355 the environment variables used to be “yes sayers” when set, this was conceptually generalized to “automatic answerers” and they just give their value as answer (as if you typed in that value when being asked). See the “usage” / “Environment Variables” section of the docs for details.
- change the builtin default for –chunker-params, create 2MiB chunks, #343 –chunker-params new default: 19,23,21,4095 - old default: 10,23,16,4095

  one of the biggest issues with borg < 1.0 (and also attic) was that it had a default target chunk size of 64kiB, thus it created a lot of chunks and thus also a huge chunk management overhead (high RAM and disk usage).

  please note that the new default won’t change the chunks that you already have in your repository. the new big chunks do not deduplicate with the old small chunks, so expect your repo to grow at least by the size of every changed file and in the worst case (e.g. if your files cache was lost / is not used) by the size of every file (minus any compression you might use).

  in case you want to immediately see a much lower resource usage (RAM / disk) for chunks management, it might be better to start with a new repo than continuing in the existing repo (with an existing repo, you’ld have to wait until all archives with small chunks got pruned to see a lower resource usage).

  if you used the old –chunker-params default value (or if you did not use –chunker-params option at all) and you’ld like to continue using small chunks (and you accept the huge resource usage that comes with that), just explicitly use borg create –chunker-params=10,23,16,4095.
- archive timestamps: the ‘time’ timestamp now refers to archive creation start time (was: end time), the new ‘time_end’ timestamp refers to archive creation end time. This might affect prune if your backups take rather long. if you give a timestamp via cli this is stored into ‘time’, therefore it now needs to mean archive creation start time.

New features:

- implement password roundtrip, #695

Bug fixes:

- remote end does not need cache nor keys directories, do not create them, #701
- added retry counter for passwords, #703

Other changes:

- fix compiler warnings, #697
- docs:
  - update README.rst to new changelog location in docs/changes.rst
  - add Teemu to AUTHORS
  - changes.rst: fix old chunker params, #698
  - FAQ: how to limit bandwidth

Version 1.0.0rc2 (2016-02-28)

New features:

- format options for location: user, pid, fqdn, hostname, now, utcnow, user

- borg list –list-format
- borg prune -v –list enables the keep/prune list output, #658

Bug fixes:

- fix _open_rb noatime handling, #657
- add a simple archivename validator, #680
- borg create –stats: show timestamps in localtime, use same labels/formatting as borg info, #651
- llfuse compatibility fixes (now compatible with: 0.40, 0.41, 0.42)

Other changes:

- it is now possible to use "pip install borgbackup[fuse]" to automatically install the llfuse dependency using the correct version requirement for it. you still need to care about having installed the FUSE / build related OS package first, though, so that building llfuse can succeed.
- Vagrant: drop Ubuntu Precise (12.04) - does not have Python >= 3.4
- Vagrant: use pyinstaller v3.1.1 to build binaries
- docs:
  - borg upgrade: add to docs that only LOCAL repos are supported
  - borg upgrade also handles borg 0.xx -> 1.0
  - use pip extras or requirements file to install llfuse
  - fix order in release process
  - updated usage docs and other minor / cosmetic fixes
  - verified borg examples in docs, #644
  - freebsd dependency installation and fuse configuration, #649
  - add example how to restore a raw device, #671
  - add a hint about the dev headers needed when installing from source
  - add examples for delete (and handle delete after list, before prune), #656
  - update example for borg create -v –stats (use iso datetime format), #663
  - added example to BORG_RSH docs
  - "connection closed by remote": add FAQ entry and point to issue #636

Version 1.0.0rc1 (2016-02-07)

New features:

- borg migrate-to-repokey ("passphrase" -> "repokey" encryption key mode)
- implement –short for borg list REPO, #611
- implement –list for borg extract (consistency with borg create)
- borg serve: overwrite client's –restrict-to-path with ssh forced command's option value (but keep everything else from the client commandline), #544
- use $XDG_CONFIG_HOME/keys for keyfile keys (~/.config/borg/keys), #515
- "borg upgrade" moves the keyfile keys to the new location

- display both archive creation start and end time in "borg info", #627

Bug fixes:

- normalize trailing slashes for the repository path, #606
- Cache: fix exception handling in __init__, release lock, #610

Other changes:

- suppress unneeded exception context (PEP 409), simpler tracebacks
- removed special code needed to deal with imperfections / incompatibilities / missing stuff in py 3.2/3.3, simplify code that can be done simpler in 3.4
- removed some version requirements that were kept on old versions because newer did not support py 3.2 any more
- use some py 3.4+ stdlib code instead of own/openssl/pypi code:
  - use os.urandom instead of own cython openssl RAND_bytes wrapper, #493
  - use hashlib.pbkdf2_hmac from py stdlib instead of own openssl wrapper
  - use hmac.compare_digest instead of == operator (constant time comparison)
  - use stat.filemode instead of homegrown code
  - use "mock" library from stdlib, #145
  - remove borg.support (with non-broken argparse copy), it is ok in 3.4+, #358
- Vagrant: copy CHANGES.rst as symlink, #592
- cosmetic code cleanups, add flake8 to tox/travis, #4
- docs / help:
  - make "borg -h" output prettier, #591
  - slightly rephrase prune help
  - add missing example for –list option of borg create
  - quote exclude line that includes an asterisk to prevent shell expansion
  - fix dead link to license
  - delete Ubuntu Vivid, it is not supported anymore (EOL)
  - OS X binary does not work for older OS X releases, #629
  - borg serve's special support for forced/original ssh commands, #544
  - misc. updates and fixes

Version 0.30.0 (2016-01-23)

Compatibility notes:

- you may need to use -v (or –info) more often to actually see output emitted at INFO log level (because it is suppressed at the default WARNING log level). See the "general" section in the usage docs.
- for borg create, you need –list (additionally to -v) to see the long file list (was needed so you can have e.g. –stats alone without the long list)

- see below about BORG_DELETE_I_KNOW_WHAT_I_AM_DOING (was: BORG_CHECK_I_KNOW_WHAT_I_AM_DOING)

Bug fixes:

- fix crash when using borg create –dry-run –keep-tag-files, #570
- make sure teardown with cleanup happens for Cache and RepositoryCache, avoiding leftover locks and TEMP dir contents, #285 (partially), #548
- fix locking KeyError, partial fix for #502
- log stats consistently, #526
- add abbreviated weekday to timestamp format, fixes #496
- strip whitespace when loading exclusions from file
- unset LD_LIBRARY_PATH before invoking ssh, fixes strange OpenSSL library version warning when using the borg binary, #514
- add some error handling/fallback for C library loading, #494
- added BORG_DELETE_I_KNOW_WHAT_I_AM_DOING for check in "borg delete", #503
- remove unused "repair" rpc method name

New features:

- borg create: implement exclusions using regular expression patterns.
- borg create: implement inclusions using patterns.
- borg extract: support patterns, #361
- support different styles for patterns:
  - fnmatch (*fm:* prefix, default when omitted), like borg <= 0.29.
  - shell (*sh:* prefix) with * not matching directory separators and **/ matching 0..n directories
  - path prefix (*pp:* prefix, for unifying borg create pp1 pp2 into the patterns system), semantics like in borg <= 0.29
  - regular expression (*re:*), new!
- –progress option for borg upgrade (#291) and borg delete <archive>
- update progress indication more often (e.g. for borg create within big files or for borg check repo), #500
- finer chunker granularity for items metadata stream, #547, #487
- borg create –list now used (additionally to -v) to enable the verbose file list output
- display borg version below tracebacks, #532

Other changes:

- hashtable size (and thus: RAM and disk consumption) follows a growth policy: grows fast while small, grows slower when getting bigger, #527
- Vagrantfile: use pyinstaller 3.1 to build binaries, freebsd sqlite3 fix, fixes #569
- no separate binaries for centos6 any more because the generic linux binaries also work on centos6 (or in general: on systems with a slightly older glibc than debian7
- dev environment: require virtualenv<14.0 so we get a py32 compatible pip
- docs:

- add space-saving chunks.archive.d trick to FAQ
- important: clarify -v and log levels in usage -> general, please read!
- sphinx configuration: create a simple man page from usage docs
- add a repo server setup example
- disable unneeded SSH features in authorized_keys examples for security.
- borg prune only knows "–keep-within" and not "–within"
- add gource video to resources docs, #507
- add netbsd install instructions
- authors: make it more clear what refers to borg and what to attic
- document standalone binary requirements, #499
- rephrase the mailing list section
- development docs: run build_api and build_usage before tagging release
- internals docs: hash table max. load factor is 0.75 now
- markup, typo, grammar, phrasing, clarifications and other fixes.
- add gcc gcc-c++ to redhat/fedora/corora install docs, fixes #583

Version 0.29.0 (2015-12-13)

Compatibility notes:

- when upgrading to 0.29.0 you need to upgrade client as well as server installations due to the locking and commandline interface changes otherwise you'll get an error msg about a RPC protocol mismatch or a wrong commandline option. if you run a server that needs to support both old and new clients, it is suggested that you have a "borg-0.28.2" and a "borg-0.29.0" command. clients then can choose via e.g. "borg –remote-path=borg-0.29.0 ...".
- the default waiting time for a lock changed from infinity to 1 second for a better interactive user experience. if the repo you want to access is currently locked, borg will now terminate after 1s with an error message. if you have scripts that shall wait for the lock for a longer time, use –lock-wait N (with N being the maximum wait time in seconds).

Bug fixes:

- hash table tuning (better chosen hashtable load factor 0.75 and prime initial size of 1031 gave ~1000x speedup in some scenarios)
- avoid creation of an orphan lock for one case, #285
- –keep-tag-files: fix file mode and multiple tag files in one directory, #432
- fixes for "borg upgrade" (attic repo converter), #466
- remove –progress isatty magic (and also –no-progress option) again, #476
- borg init: display proper repo URL
- fix format of umask in help pages, #463

New features:

- implement –lock-wait, support timeout for UpgradableLock, #210

- implement borg break-lock command, #157
- include system info below traceback, #324
- sane remote logging, remote stderr, #461:
  - remote log output: intercept it and log it via local logging system, with "Remote: " prefixed to message. log remote tracebacks.
  - remote stderr: output it to local stderr with "Remote: " prefixed.
- add –debug and –info (same as –verbose) to set the log level of the builtin logging configuration (which otherwise defaults to warning), #426 note: there are few messages emitted at DEBUG level currently.
- optionally configure logging via env var BORG_LOGGING_CONF
- add –filter option for status characters: e.g. to show only the added or modified files (and also errors), use "borg create -v –filter=AME ...".
- more progress indicators, #394
- use ISO-8601 date and time format, #375
- "borg check –prefix" to restrict archive checking to that name prefix, #206

Other changes:

- hashindex_add C implementation (speed up cache re-sync for new archives)
- increase FUSE read_size to 1024 (speed up metadata operations)
- check/delete/prune –save-space: free unused segments quickly, #239
- increase rpc protocol version to 2 (see also Compatibility notes), #458
- silence borg by default (via default log level WARNING)
- get rid of C compiler warnings, #391
- upgrade OS X FUSE to 3.0.9 on the OS X binary build system
- use python 3.5.1 to build binaries
- docs:
  - new mailing list borgbackup@python.org, #468
  - readthedocs: color and logo improvements
  - load coverage icons over SSL (avoids mixed content)
  - more precise binary installation steps
  - update release procedure docs about OS X FUSE
  - FAQ entry about unexpected 'A' status for unchanged file(s), #403
  - add docs about 'E' file status
  - add "borg upgrade" docs, #464
  - add developer docs about output and logging
  - clarify encryption, add note about client-side encryption
  - add resources section, with videos, talks, presentations, #149
  - Borg moved to Arch Linux [community]
  - fix wrong installation instructions for archlinux

Version 0.28.2 (2015-11-15)

New features:

- borg create –exclude-if-present TAGFILE - exclude directories that have the given file from the backup. You can additionally give –keep-tag-files to preserve just the directory roots and the tag-files (but not backup other directory contents), #395, attic #128, attic #142

Other changes:

- do not create docs sources at build time (just have them in the repo), completely remove have_cython() hack, do not use the "mock" library at build time, #384
- avoid hidden import, make it easier for PyInstaller, easier fix for #218
- docs:
  - add description of item flags / status output, fixes #402
  - explain how to regenerate usage and API files (build_api or build_usage) and when to commit usage files directly into git, #384
  - minor install docs improvements

Version 0.28.1 (2015-11-08)

Bug fixes:

- do not try to build api / usage docs for production install, fixes unexpected "mock" build dependency, #384

Other changes:

- avoid using msgpack.packb at import time
- fix formatting issue in changes.rst
- fix build on readthedocs

Version 0.28.0 (2015-11-08)

Compatibility notes:

- changed return codes (exit codes), see docs. in short: old: 0 = ok, 1 = error. now: 0 = ok, 1 = warning, 2 = error

New features:

- refactor return codes (exit codes), fixes #61
- add –show-rc option enable "terminating with X status, rc N" output, fixes 58, #351
- borg create backups atime and ctime additionally to mtime, fixes #317 - extract: support atime additionally to mtime - FUSE: support ctime and atime additionally to mtime
- support borg –version
- emit a warning if we have a slow msgpack installed
- borg list –prefix=thishostname- REPO, fixes #205
- Debug commands (do not use except if you know what you do: debug-get-obj, debug-put-obj, debug-delete-obj, debug-dump-archive-items.

Bug fixes:

- setup.py: fix bug related to BORG_LZ4_PREFIX processing
- fix "check" for repos that have incomplete chunks, fixes #364
- borg mount: fix unlocking of repository at umount time, fixes #331
- fix reading files without touching their atime, #334
- non-ascii ACL fixes for Linux, FreeBSD and OS X, #277
- fix acl_use_local_uid_gid() and add a test for it, attic #359
- borg upgrade: do not upgrade repositories in place by default, #299
- fix cascading failure with the index conversion code, #269
- borg check: implement 'cmdline' archive metadata value decoding, #311
- fix RobustUnpacker, it missed some metadata keys (new atime and ctime keys were missing, but also bsdflags). add check for unknown metadata keys.
- create from stdin: also save atime, ctime (cosmetic)
- use default_notty=False for confirmations, fixes #345
- vagrant: fix msgpack installation on centos, fixes #342
- deal with unicode errors for symlinks in same way as for regular files and have a helpful warning message about how to fix wrong locale setup, fixes #382
- add ACL keys the RobustUnpacker must know about

Other changes:

- improve file size displays, more flexible size formatters
- explicitly commit to the units standard, #289
- archiver: add E status (means that an error occurred when processing this (single) item
- do binary releases via "github releases", closes #214
- create: use -x and –one-file-system (was: –do-not-cross-mountpoints), #296
- a lot of changes related to using "logging" module and screen output, #233
- show progress display if on a tty, output more progress information, #303
- factor out status output so it is consistent, fix surrogates removal, maybe fixes #309
- move away from RawConfigParser to ConfigParser
- archive checker: better error logging, give chunk_id and sequence numbers (can be used together with borg debug-dump-archive-items).
- do not mention the deprecated passphrase mode
- emit a deprecation warning for –compression N (giving a just a number)
- misc .coverragerc fixes (and coverage measurement improvements), fixes #319
- refactor confirmation code, reduce code duplication, add tests
- prettier error messages, fixes #307, #57
- tests:
  - add a test to find disk-full issues, #327
  - travis: also run tests on Python 3.5

  - travis: use tox -r so it rebuilds the tox environments
  - test the generated pyinstaller-based binary by archiver unit tests, #215
  - vagrant: tests: announce whether fakeroot is used or not
  - vagrant: add vagrant user to fuse group for debianoid systems also
  - vagrant: llfuse install on darwin needs pkgconfig installed
  - vagrant: use pyinstaller from develop branch, fixes #336
  - benchmarks: test create, extract, list, delete, info, check, help, fixes #146
  - benchmarks: test with both the binary and the python code
  - archiver tests: test with both the binary and the python code, fixes #215
  - make basic test more robust
- docs:
  - moved docs to borgbackup.readthedocs.org, #155
  - a lot of fixes and improvements, use mobile-friendly RTD standard theme
  - use zlib,6 compression in some examples, fixes #275
  - add missing rename usage to docs, closes #279
  - include the help offered by borg help <topic> in the usage docs, fixes #293
  - include a list of major changes compared to attic into README, fixes #224
  - add OS X install instructions, #197
  - more details about the release process, #260
  - fix linux glibc requirement (binaries built on debian7 now)
  - build: move usage and API generation to setup.py
  - update docs about return codes, #61
  - remove api docs (too much breakage on rtd)
  - borgbackup install + basics presentation (asciinema)
  - describe the current style guide in documentation
  - add section about debug commands
  - warn about not running out of space
  - add example for rename
  - improve chunker params docs, fixes #362
  - minor development docs update

Version 0.27.0 (2015-10-07)

New features:

- "borg upgrade" command - attic -> borg one time converter / migration, #21
- temporary hack to avoid using lots of disk space for chunks.archive.d, #235: To use it: rm -rf chunks.archive.d ; touch chunks.archive.d

- respect XDG_CACHE_HOME, attic #181
- add support for arbitrary SSH commands, attic #99
- borg delete –cache-only REPO (only delete cache, not REPO), attic #123

Bug fixes:

- use Debian 7 (wheezy) to build pyinstaller borgbackup binaries, fixes slow down observed when running the Centos6-built binary on Ubuntu, #222
- do not crash on empty lock.roster, fixes #232
- fix multiple issues with the cache config version check, #234
- fix segment entry header size check, attic #352 plus other error handling improvements / code deduplication there.
- always give segment and offset in repo IntegrityErrors

Other changes:

- stop producing binary wheels, remove docs about it, #147
- docs: - add warning about prune - generate usage include files only as needed - development docs: add Vagrant section - update / improve / reformat FAQ - hint to single-file pyinstaller binaries from README

Version 0.26.1 (2015-09-28)

This is a minor update, just docs and new pyinstaller binaries.

- docs update about python and binary requirements
- better docs for –read-special, fix #220
- re-built the binaries, fix #218 and #213 (glibc version issue)
- update web site about single-file pyinstaller binaries

Note: if you did a python-based installation, there is no need to upgrade.

Version 0.26.0 (2015-09-19)

New features:

- Faster cache sync (do all in one pass, remove tar/compression stuff), #163
- BORG_REPO env var to specify the default repo, #168
- read special files as if they were regular files, #79
- implement borg create –dry-run, attic issue #267
- Normalize paths before pattern matching on OS X, #143
- support OpenBSD and NetBSD (except xattrs/ACLs)
- support / run tests on Python 3.5

Bug fixes:

- borg mount repo: use absolute path, attic #200, attic #137
- chunker: use off_t to get 64bit on 32bit platform, #178
- initialize chunker fd to -1, so it's not equal to STDIN_FILENO (0)

- fix reaction to "no" answer at delete repo prompt, #182
- setup.py: detect lz4.h header file location
- to support python < 3.2.4, add less buggy argparse lib from 3.2.6 (#194)
- fix for obtaining `char *` from temporary Python value (old code causes a compile error on Mint 17.2)
- llfuse 0.41 install troubles on some platforms, require < 0.41 (UnicodeDecodeError exception due to non-ascii llfuse setup.py)
- cython code: add some int types to get rid of unspecific python add / subtract operations (avoid `undefined symbol FPE_`... error on some platforms)
- fix verbose mode display of stdin backup
- extract: warn if a include pattern never matched, fixes #209, implement counters for Include/ExcludePatterns
- archive names with slashes are invalid, attic issue #180
- chunker: add a check whether the POSIX_FADV_DONTNEED constant is defined - fixes building on OpenBSD.

Other changes:

- detect inconsistency / corruption / hash collision, #170
- replace versioneer with setuptools_scm, #106
- docs:
  - pkg-config is needed for llfuse installation
  - be more clear about pruning, attic issue #132
- unit tests:
  - xattr: ignore security.selinux attribute showing up
  - ext3 seems to need a bit more space for a sparse file
  - do not test lzma level 9 compression (avoid MemoryError)
  - work around strange mtime granularity issue on netbsd, fixes #204
  - ignore st_rdev if file is not a block/char device, fixes #203
  - stay away from the setgid and sticky mode bits
- use Vagrant to do easy cross-platform testing (#196), currently:
  - Debian 7 "wheezy" 32bit, Debian 8 "jessie" 64bit
  - Ubuntu 12.04 32bit, Ubuntu 14.04 64bit
  - Centos 7 64bit
  - FreeBSD 10.2 64bit
  - OpenBSD 5.7 64bit
  - NetBSD 6.1.5 64bit
  - Darwin (OS X Yosemite)

Version 0.25.0 (2015-08-29)

Compatibility notes:

- lz4 compression library (liblz4) is a new requirement (#156)
- the new compression code is very compatible: as long as you stay with zlib compression, older borg releases will still be able to read data from a repo/archive made with the new code (note: this is not the case for the default "none" compression, use "zlib,0" if you want a "no compression" mode that can be read by older borg). Also the new code is able to read repos and archives made with older borg versions (for all zlib levels 0..9).

Deprecations:

- –compression N (with N being a number, as in 0.24) is deprecated. We keep the –compression 0..9 for now to not break scripts, but it is deprecated and will be removed later, so better fix your scripts now: –compression 0 (as in 0.24) is the same as –compression zlib,0 (now). BUT: if you do not want compression, you rather want –compression none (which is the default). –compression 1 (in 0.24) is the same as –compression zlib,1 (now) –compression 9 (in 0.24) is the same as –compression zlib,9 (now)

New features:

- create –compression none (default, means: do not compress, just pass through data "as is". this is more efficient than zlib level 0 as used in borg 0.24)
- create –compression lz4 (super-fast, but not very high compression)
- create –compression zlib,N (slower, higher compression, default for N is 6)
- create –compression lzma,N (slowest, highest compression, default N is 6)
- honor the nodump flag (UF_NODUMP) and do not backup such items
- list –short just outputs a simple list of the files/directories in an archive

Bug fixes:

- fixed –chunker-params parameter order confusion / malfunction, fixes #154
- close fds of segments we delete (during compaction)
- close files which fell out the lrucache
- fadvise DONTNEED now is only called for the byte range actually read, not for the whole file, fixes #158.
- fix issue with negative "all archives" size, fixes #165
- restore_xattrs: ignore if setxattr fails with EACCES, fixes #162

Other changes:

- remove fakeroot requirement for tests, tests run faster without fakeroot (test setup does not fail any more without fakeroot, so you can run with or without fakeroot), fixes #151 and #91.
- more tests for archiver
- recover_segment(): don't assume we have an fd for segment
- lrucache refactoring / cleanup, add dispose function, py.test tests
- generalize hashindex code for any key length (less hardcoding)
- lock roster: catch file not found in remove() method and ignore it
- travis CI: use requirements file
- improved docs:
  - replace hack for llfuse with proper solution (install libfuse-dev)

- update docs about compression
- update development docs about fakeroot
- internals: add some words about lock files / locking system
- support: mention BountySource and for what it can be used
- theme: use a lighter green
- add pypi, wheel, dist package based install docs
- split install docs into system-specific preparations and generic instructions

Version 0.24.0 (2015-08-09)

Incompatible changes (compared to 0.23):

- borg now always issues –umask NNN option when invoking another borg via ssh on the repository server. By that, it's making sure it uses the same umask for remote repos as for local ones. Because of this, you must upgrade both server and client(s) to 0.24.
- the default umask is 077 now (if you do not specify via –umask) which might be a different one as you used previously. The default umask avoids that you accidentally give access permissions for group and/or others to files created by borg (e.g. the repository).

Deprecations:

- "–encryption passphrase" mode is deprecated, see #85 and #97. See the new "–encryption repokey" mode for a replacement.

New features:

- borg create –chunker-params ... to configure the chunker, fixes #16 (attic #302, attic #300, and somehow also #41). This can be used to reduce memory usage caused by chunk management overhead, so borg does not create a huge chunks index/repo index and eats all your RAM if you back up lots of data in huge files (like VM disk images). See docs/misc/create_chunker-params.txt for more information.
- borg info now reports chunk counts in the chunk index.
- borg create –compression 0..9 to select zlib compression level, fixes #66 (attic #295).
- borg init –encryption repokey (to store the encryption key into the repo), fixes #85
- improve at-end error logging, always log exceptions and set exit_code=1
- LoggedIO: better error checks / exceptions / exception handling
- implement –remote-path to allow non-default-path borg locations, #125
- implement –umask M and use 077 as default umask for better security, #117
- borg check: give a named single archive to it, fixes #139
- cache sync: show progress indication
- cache sync: reimplement the chunk index merging in C

Bug fixes:

- fix segfault that happened for unreadable files (chunker: n needs to be a signed size_t), #116
- fix the repair mode, #144
- repo delete: add destroy to allowed rpc methods, fixes issue #114

- more compatible repository locking code (based on mkdir), maybe fixes #92 (attic #317, attic #201).
- better Exception msg if no Borg is installed on the remote repo server, #56
- create a RepositoryCache implementation that can cope with >2GiB, fixes attic #326.
- fix Traceback when running check –repair, attic #232
- clarify help text, fixes #73.
- add help string for –no-files-cache, fixes #140

Other changes:

- improved docs:
  - added docs/misc directory for misc. writeups that won't be included "as is" into the html docs.
  - document environment variables and return codes (attic #324, attic #52)
  - web site: add related projects, fix web site url, IRC #borgbackup
  - Fedora/Fedora-based install instructions added to docs
  - Cygwin-based install instructions added to docs
  - updated AUTHORS
  - add FAQ entries about redundancy / integrity
  - clarify that borg extract uses the cwd as extraction target
  - update internals doc about chunker params, memory usage and compression
  - added docs about development
  - add some words about resource usage in general
  - document how to backup a raw disk
  - add note about how to run borg from virtual env
  - add solutions for (ll)fuse installation problems
  - document what borg check does, fixes #138
  - reorganize borgbackup.github.io sidebar, prev/next at top
  - deduplicate and refactor the docs / README.rst
- use borg-tmp as prefix for temporary files / directories
- short prune options without "keep-" are deprecated, do not suggest them
- improved tox configuration
- remove usage of unittest.mock, always use mock from pypi
- use entrypoints instead of scripts, for better use of the wheel format and modern installs
- add requirements.d/development.txt and modify tox.ini
- use travis-ci for testing based on Linux and (new) OS X
- use coverage.py, pytest-cov and codecov.io for test coverage support

I forgot to list some stuff already implemented in 0.23.0, here they are:

New features:

- efficient archive list from manifest, meaning a big speedup for slow repo connections and "list <repo>", "delete <repo>", "prune" (attic #242, attic #167)
- big speedup for chunks cache sync (esp. for slow repo connections), fixes #18
- hashindex: improve error messages

Other changes:

- explicitly specify binary mode to open binary files
- some easy micro optimizations

Version 0.23.0 (2015-06-11)

Incompatible changes (compared to attic, fork related):

- changed sw name and cli command to "borg", updated docs
- package name (and name in urls) uses "borgbackup" to have less collisions
- changed repo / cache internal magic strings from ATTIC* to BORG*, changed cache location to .cache/borg/ - this means that it currently won't accept attic repos (see issue #21 about improving that)

Bug fixes:

- avoid defect python-msgpack releases, fixes attic #171, fixes attic #185
- fix traceback when trying to do unsupported passphrase change, fixes attic #189
- datetime does not like the year 10.000, fixes attic #139
- fix "info" all archives stats, fixes attic #183
- fix parsing with missing microseconds, fixes attic #282
- fix misleading hint the fuse ImportError handler gave, fixes attic #237
- check unpacked data from RPC for tuple type and correct length, fixes attic #127
- fix Repository._active_txn state when lock upgrade fails
- give specific path to xattr.is_enabled(), disable symlink setattr call that always fails
- fix test setup for 32bit platforms, partial fix for attic #196
- upgraded versioneer, PEP440 compliance, fixes attic #257

New features:

- less memory usage: add global option –no-cache-files
- check –last N (only check the last N archives)
- check: sort archives in reverse time order
- rename repo::oldname newname (rename repository)
- create -v output more informative
- create –progress (backup progress indicator)
- create –timestamp (utc string or reference file/dir)
- create: if "-" is given as path, read binary from stdin
- extract: if –stdout is given, write all extracted binary data to stdout

- extract –sparse (simple sparse file support)
- extra debug information for 'fread failed'
- delete <repo> (deletes whole repo + local cache)
- FUSE: reflect deduplication in allocated blocks
- only allow whitelisted RPC calls in server mode
- normalize source/exclude paths before matching
- use posix_fadvise to not spoil the OS cache, fixes attic #252
- toplevel error handler: show tracebacks for better error analysis
- sigusr1 / sigint handler to print current file infos - attic PR #286
- RPCError: include the exception args we get from remote

Other changes:

- source: misc. cleanups, pep8, style
- docs and faq improvements, fixes, updates
- cleanup crypto.pyx, make it easier to adapt to other AES modes
- do os.fsync like recommended in the python docs
- source: Let chunker optionally work with os-level file descriptor.
- source: Linux: remove duplicate os.fsencode calls
- source: refactor _open_rb code a bit, so it is more consistent / regular
- source: refactor indicator (status) and item processing
- source: use py.test for better testing, flake8 for code style checks
- source: fix tox >=2.0 compatibility (test runner)
- pypi package: add python version classifiers, add FreeBSD to platforms

Attic Changelog

Here you can see the full list of changes between each Attic release until Borg forked from Attic:

Version 0.17 (bugfix release, released on X)

- Fix hashindex ARM memory alignment issue (#309)
- Improve hashindex error messages (#298)

Version 0.16 (bugfix release, released on May 16, 2015)

- Fix typo preventing the security confirmation prompt from working (#303)
- Improve handling of systems with improperly configured file system encoding (#289)
- Fix "All archives" output for attic info. (#183)
- More user friendly error message when repository key file is not found (#236)
- Fix parsing of iso 8601 timestamps with zero microseconds (#282)

Version 0.15 (bugfix release, released on Apr 15, 2015)

- xattr: Be less strict about unknown/unsupported platforms (#239)
- Reduce repository listing memory usage (#163).
- Fix BrokenPipeError for remote repositories (#233)
- Fix incorrect behavior with two character directory names (#265, #268)
- Require approval before accessing relocated/moved repository (#271)
- Require approval before accessing previously unknown unencrypted repositories (#271)
- Fix issue with hash index files larger than 2GB.
- Fix Python 3.2 compatibility issue with noatime open() (#164)
- Include missing pyx files in dist files (#168)

Version 0.14 (feature release, released on Dec 17, 2014)

- Added support for stripping leading path segments (#95) "attic extract –strip-segments X"
- Add workaround for old Linux systems without acl_extended_file_no_follow (#96)
- Add MacPorts' path to the default openssl search path (#101)
- HashIndex improvements, eliminates unnecessary IO on low memory systems.
- Fix "Number of files" output for attic info. (#124)
- limit create file permissions so files aren't read while restoring
- Fix issue with empty xattr values (#106)

Version 0.13 (feature release, released on Jun 29, 2014)

- Fix sporadic "Resource temporarily unavailable" when using remote repositories
- Reduce file cache memory usage (#90)
- Faster AES encryption (utilizing AES-NI when available)
- Experimental Linux, OS X and FreeBSD ACL support (#66)
- Added support for backup and restore of BSDFlags (OSX, FreeBSD) (#56)
- Fix bug where xattrs on symlinks were not correctly restored
- Added cachedir support. CACHEDIR.TAG compatible cache directories can now be excluded using `--exclude-caches` (#74)
- Fix crash on extreme mtime timestamps (year 2400+) (#81)
- Fix Python 3.2 specific lockf issue (EDEADLK)

Version 0.12 (feature release, released on April 7, 2014)

- Python 3.4 support (#62)
- Various documentation improvements a new style
- `attic mount` now supports mounting an entire repository not only individual archives (#59)

- Added option to restrict remote repository access to specific path(s): `attic serve --restrict-to-path X` (#51)
- Include "all archives" size information in "–stats" output. (#54)
- Added `--stats` option to `attic delete` and `attic prune`
- Fixed bug where `attic prune` used UTC instead of the local time zone when determining which archives to keep.
- Switch to SI units (Power of 1000 instead 1024) when printing file sizes

Version 0.11 (feature release, released on March 7, 2014)

- New "check" command for repository consistency checking (#24)
- Documentation improvements
- Fix exception during "attic create" with repeated files (#39)
- New "–exclude-from" option for attic create/extract/verify.
- Improved archive metadata deduplication.
- "attic verify" has been deprecated. Use "attic extract –dry-run" instead.
- "attic prune –hourly|daily|..." has been deprecated. Use "attic prune –keep-hourly|daily|..." instead.
- Ignore xattr errors during "extract" if not supported by the filesystem. (#46)

Version 0.10 (bugfix release, released on Jan 30, 2014)

- Fix deadlock when extracting 0 sized files from remote repositories
- "–exclude" wildcard patterns are now properly applied to the full path not just the file name part (#5).
- Make source code endianness agnostic (#1)

Version 0.9 (feature release, released on Jan 23, 2014)

- Remote repository speed and reliability improvements.
- Fix sorting of segment names to ignore NFS left over files. (#17)
- Fix incorrect display of time (#13)
- Improved error handling / reporting. (#12)
- Use fcntl() instead of flock() when locking repository/cache. (#15)
- Let ssh figure out port/user if not specified so we don't override .ssh/config (#9)
- Improved libcrypto path detection (#23).

Version 0.8.1 (bugfix release, released on Oct 4, 2013)

- Fix segmentation fault issue.

Version 0.8 (feature release, released on Oct 3, 2013)

- Fix xattr issue when backing up sshfs filesystems (#4)
- Fix issue with excessive index file size (#6)
- Support access of read only repositories.
- **New syntax to enable repository encryption:** attic init –encryption="none|passphrase|keyfile".
- Detect and abort if repository is older than the cache.

Version 0.7 (feature release, released on Aug 5, 2013)

- Ported to FreeBSD
- Improved documentation
- Experimental: Archives mountable as fuse filesystems.
- The "user." prefix is no longer stripped from xattrs on Linux

Version 0.6.1 (bugfix release, released on July 19, 2013)

- Fixed an issue where mtime was not always correctly restored.

Version 0.6 First public release on July 9, 2013

Internals

This page documents the internal data structures and storage mechanisms of Borg. It is partly based on mailing list discussion about internals and also on static code analysis.

Repository and Archives

Borg stores its data in a *Repository*. Each repository can hold multiple *Archives*, which represent individual backups that contain a full archive of the files specified when the backup was performed. Deduplication is performed across multiple backups, both on data and metadata, using *Chunks* created by the chunker using the Buzhash algorithm.

Each repository has the following file structure:

README simple text file telling that this is a Borg repository

config repository configuration

data/ directory where the actual data is stored

hints.%d hints for repository compaction

index.%d repository index

lock.roster and lock.exclusive/* used by the locking system to manage shared and exclusive locks

Lock files

Borg uses locks to get (exclusive or shared) access to the cache and the repository.

The locking system is based on creating a directory *lock.exclusive* (for exclusive locks). Inside the lock directory, there is a file indication hostname, process id and thread id of the lock holder.

There is also a json file *lock.roster* that keeps a directory of all shared and exclusive lockers.

If the process can create the *lock.exclusive* directory for a resource, it has the lock for it. If creation fails (because the directory has already been created by some other process), lock acquisition fails.

The cache lock is usually in *~/.cache/borg/REPOID/lock.**. The repository lock is in *repository/lock.**.

In case you run into troubles with the locks, you can use the `borg break-lock` command after you first have made sure that no Borg process is running on any machine that accesses this resource. Be very careful, the cache or repository might get damaged if multiple processes use it at the same time.

Config file

Each repository has a `config` file which which is a `INI`-style file and looks like this:

```
[repository]
version = 1
segments_per_dir = 10000
max_segment_size = 5242880
id = 57d6c1d52ce76a836b532b0e42e677dec6af9fca3673db511279358828a21ed6
```

This is where the `repository.id` is stored. It is a unique identifier for repositories. It will not change if you move the repository around so you can make a local transfer then decide to move the repository to another (even remote) location at a later time.

Keys

The key to address the key/value store is usually computed like this:

key = id = id_hash(unencrypted_data)

The id_hash function is:

- sha256 (no encryption keys available)
- hmac-sha256 (encryption keys available)

Segments and archives

A Borg repository is a filesystem based transactional key/value store. It makes extensive use of msgpack to store data and, unless otherwise noted, data is stored in msgpack encoded files.

Objects referenced by a key are stored inline in files (*segments*) of approx. 5MB size in numbered subdirectories of `repo/data`.

They contain:

- header size
- crc
- size

- tag
- key
- data

Segments are built locally, and then uploaded. Those files are strictly append-only and modified only once.

Tag is either `PUT`, `DELETE`, or `COMMIT`. A segment file is basically a transaction log where each repository operation is appended to the file. So if an object is written to the repository a `PUT` tag is written to the file followed by the object id and data. If an object is deleted a `DELETE` tag is appended followed by the object id. A `COMMIT` tag is written when a repository transaction is committed. When a repository is opened any `PUT` or `DELETE` operations not followed by a `COMMIT` tag are discarded since they are part of a partial/uncommitted transaction.

The manifest

The manifest is an object with an all-zero key that references all the archives. It contains:

- version
- list of archive infos
- timestamp
- config

Each archive info contains:

- name
- id
- time

It is the last object stored, in the last segment, and is replaced each time.

The Archive

The archive metadata does not contain the file items directly. Only references to other objects that contain that data. An archive is an object that contains:

- version
- name
- list of chunks containing item metadata (size: count * ~40B)
- cmdline
- hostname
- username
- time

Note about archive limitations The archive is currently stored as a single object in the repository and thus limited in size to MAX_OBJECT_SIZE (20MiB).

As one chunk list entry is ~40B, that means we can reference ~500.000 item metadata stream chunks per archive.

Each item metadata stream chunk is ~128kiB (see hardcoded ITEMS_CHUNKER_PARAMS).

So that means the whole item metadata stream is limited to ~64GiB chunks. If compression is used, the amount of storable metadata is bigger - by the compression factor.

If the medium size of an item entry is 100B (small size file, no ACLs/xattrs), that means a limit of ~640 million files/directories per archive.

If the medium size of an item entry is 2kB (~100MB size files or more ACLs/xattrs), the limit will be ~32 million files/directories per archive.

If one tries to create an archive object bigger than MAX_OBJECT_SIZE, a fatal IntegrityError will be raised.

A workaround is to create multiple archives with less items each, see also #1452.

The Item

Each item represents a file, directory or other fs item and is stored as an `item` dictionary that contains:

- path
- list of data chunks (size: count * ~40B)
- user
- group
- uid
- gid
- mode (item type + permissions)
- source (for links)
- rdev (for devices)
- mtime, atime, ctime in nanoseconds
- xattrs
- acl
- bsdfiles

All items are serialized using msgpack and the resulting byte stream is fed into the same chunker algorithm as used for regular file data and turned into deduplicated chunks. The reference to these chunks is then added to the archive metadata. To achieve a finer granularity on this metadata stream, we use different chunker params for this chunker, which result in smaller chunks.

A chunk is stored as an object as well, of course.

Chunks

The Borg chunker uses a rolling hash computed by the Buzhash algorithm. It triggers (chunks) when the last HASH_MASK_BITS bits of the hash are zero, producing chunks of 2^HASH_MASK_BITS Bytes on average.

`borg create --chunker-params CHUNK_MIN_EXP,CHUNK_MAX_EXP,HASH_MASK_BITS,HASH_WINDOW_SIZE` can be used to tune the chunker parameters, the default is:

- CHUNK_MIN_EXP = 19 (minimum chunk size = 2^19 B = 512 kiB)
- CHUNK_MAX_EXP = 23 (maximum chunk size = 2^23 B = 8 MiB)
- HASH_MASK_BITS = 21 (statistical medium chunk size ~= 2^21 B = 2 MiB)

- HASH_WINDOW_SIZE = 4095 [B] (*0xFFF*)

The buzhash table is altered by XORing it with a seed randomly generated once for the archive, and stored encrypted in the keyfile. This is to prevent chunk size based fingerprinting attacks on your encrypted repo contents (to guess what files you have based on a specific set of chunk sizes).

For some more general usage hints see also `--chunker-params`.

Indexes / Caches

The **files cache** is stored in `cache/files` and is indexed on the `file path hash`. At backup time, it is used to quickly determine whether we need to chunk a given file (or whether it is unchanged and we already have all its pieces). It contains:

- age
- file inode number
- file size
- file mtime_ns
- file content chunk hashes

The inode number is stored to make sure we distinguish between different files, as a single path may not be unique across different archives in different setups.

The files cache is stored as a python associative array storing python objects, which generates a lot of overhead.

The **chunks cache** is stored in `cache/chunks` and is indexed on the `chunk id_hash`. It is used to determine whether we already have a specific chunk, to count references to it and also for statistics. It contains:

- reference count
- size
- encrypted/compressed size

The **repository index** is stored in `repo/index.%d` and is indexed on the `chunk id_hash`. It is used to determine a chunk's location in the repository. It contains:

- segment (that contains the chunk)
- offset (where the chunk is located in the segment)

The repository index file is random access.

Hints are stored in a file (`repo/hints.%d`). It contains:

- version
- list of segments
- compact

hints and index can be recreated if damaged or lost using `check --repair`.

The chunks cache and the repository index are stored as hash tables, with only one slot per bucket, but that spreads the collisions to the following buckets. As a consequence the hash is just a start position for a linear search, and if the element is not in the table the index is linearly crossed until an empty bucket is found.

When the hash table is filled to 75%, its size is grown. When it's emptied to 25%, its size is shrinked. So operations on it have a variable complexity between constant and linear with low factor, and memory overhead varies between 33% and 300%.

Indexes / Caches memory usage

Here is the estimated memory usage of Borg:

chunk_count ~= total_file_size / 2 ^ HASH_MASK_BITS

repo_index_usage = chunk_count * 40

chunks_cache_usage = chunk_count * 44

files_cache_usage = total_file_count * 240 + chunk_count * 80

mem_usage ~= repo_index_usage + chunks_cache_usage + files_cache_usage = chunk_count * 164 + total_file_count * 240

All units are Bytes.

It is assuming every chunk is referenced exactly once (if you have a lot of duplicate chunks, you will have less chunks than estimated above).

It is also assuming that typical chunk size is 2^HASH_MASK_BITS (if you have a lot of files smaller than this statistical medium chunk size, you will have more chunks than estimated above, because 1 file is at least 1 chunk).

If a remote repository is used the repo index will be allocated on the remote side.

E.g. backing up a total count of 1 Mi (IEC binary prefix e.g. 2^20) files with a total size of 1TiB.

1. with `create --chunker-params 10,23,16,4095` (custom, like borg < 1.0 or attic):

   mem_usage = 2.8GiB

2. with `create --chunker-params 19,23,21,4095` (default):

   mem_usage = 0.31GiB

Note: There is also the `--no-files-cache` option to switch off the files cache. You'll save some memory, but it will need to read / chunk all the files as it can not skip unmodified files then.

Encryption

AES-256 is used in CTR mode (so no need for padding). A 64bit initialization vector is used, a HMAC-SHA256 is computed on the encrypted chunk with a random 64bit nonce and both are stored in the chunk. The header of each chunk is: `TYPE(1)` + `HMAC(32)` + `NONCE(8)` + `CIPHERTEXT`. Encryption and HMAC use two different keys.

In AES CTR mode you can think of the IV as the start value for the counter. The counter itself is incremented by one after each 16 byte block. The IV/counter is not required to be random but it must NEVER be reused. So to accomplish this Borg initializes the encryption counter to be higher than any previously used counter value before encrypting new data.

To reduce payload size, only 8 bytes of the 16 bytes nonce is saved in the payload, the first 8 bytes are always zeros. This does not affect security but limits the maximum repository capacity to only 295 exabytes (2**64 * 16 bytes).

Encryption keys (and other secrets) are kept either in a key file on the client ('keyfile' mode) or in the repository config on the server ('repokey' mode). In both cases, the secrets are generated from random and then encrypted by a key derived from your passphrase (this happens on the client before the key is stored into the keyfile or as repokey).

The passphrase is passed through the `BORG_PASSPHRASE` environment variable or prompted for interactive usage.

Key files

When initialized with the `init -e keyfile` command, Borg needs an associated file in `$HOME/.config/borg/keys` to read and write the repository. The format is based on msgpack, base64 encoding and PBKDF2 SHA256 hashing, which is then encoded again in a msgpack.

The internal data structure is as follows:

version currently always an integer, 1

repository_id the `id` field in the `config INI` file of the repository.

enc_key the key used to encrypt data with AES (256 bits)

enc_hmac_key the key used to HMAC the encrypted data (256 bits)

id_key the key used to HMAC the plaintext chunk data to compute the chunk's id

chunk_seed the seed for the buzhash chunking table (signed 32 bit integer)

Those fields are processed using msgpack. The utf-8 encoded passphrase is processed with PBKDF2 (SHA256, 100000 iterations, random 256 bit salt) to give us a derived key. The derived key is 256 bits long. A HMAC-SHA256 checksum of the above fields is generated with the derived key, then the derived key is also used to encrypt the above pack of fields. Then the result is stored in a another msgpack formatted as follows:

version currently always an integer, 1

salt random 256 bits salt used to process the passphrase

iterations number of iterations used to process the passphrase (currently 100000)

algorithm the hashing algorithm used to process the passphrase and do the HMAC checksum (currently the string `sha256`)

hash the HMAC of the encrypted derived key

data the derived key, encrypted with AES over a PBKDF2 SHA256 key described above

The resulting msgpack is then encoded using base64 and written to the key file, wrapped using the standard `textwrap` module with a header. The header is a single line with a MAGIC string, a space and a hexadecimal representation of the repository id.

Compression

Borg supports the following compression methods:

- none (no compression, pass through data 1:1)
- lz4 (low compression, but super fast)
- zlib (level 0-9, level 0 is no compression [but still adding zlib overhead], level 1 is low, level 9 is high compression)
- lzma (level 0-9, level 0 is low, level 9 is high compression).

Speed: none > lz4 > zlib > lzma Compression: lzma > zlib > lz4 > none

Be careful, higher zlib and especially lzma compression levels might take a lot of resources (CPU and memory).

The overall speed of course also depends on the speed of your target storage. If that is slow, using a higher compression level might yield better overall performance. You need to experiment a bit. Maybe just watch your CPU load, if that is relatively low, increase compression until 1 core is 70-100% loaded.

Even if your target storage is rather fast, you might see interesting effects: while doing no compression at all (none) is a operation that takes no time, it likely will need to store more data to the storage compared to using lz4. The time needed to transfer and store the additional data might be much more than if you had used lz4 (which is super fast, but still might compress your data about 2:1). This is assuming your data is compressible (if you backup already compressed data, trying to compress them at backup time is usually pointless).

Compression is applied after deduplication, thus using different compression methods in one repo does not influence deduplication.

See `borg create --help` about how to specify the compression level and its default.

Development

This chapter will get you started with Borg development.

Borg is written in Python (with a little bit of Cython and C for the performance critical parts).

Style guide

We generally follow pep8, with 120 columns instead of 79. We do *not* use form-feed (`^L`) characters to separate sections either. Compliance is tested automatically when you run the tests.

Continuous Integration

All pull requests go through Travis-CI, which runs the tests on Linux and Mac OS X as well as the flake8 style checker. Additional Unix-like platforms are tested on Golem.

Output and Logging

When writing logger calls, always use correct log level (debug only for debugging, info for informative messages, warning for warnings, error for errors, critical for critical errors/states).

When directly talking to the user (e.g. Y/N questions), do not use logging, but directly output to stderr (not: stdout, it could be connected to a pipe).

To control the amount and kinds of messages output to stderr or emitted at info level, use flags like `--stats` or `--list`.

Building a development environment

First, just install borg into a virtual env as described before.

To install some additional packages needed for running the tests, activate your virtual env and run:

```
pip install -r requirements.d/development.txt
```

Running the tests

The tests are in the borg/testsuite package.

To run all the tests, you need to have fakeroot installed. If you do not have fakeroot, you still will be able to run most tests, just leave away the *fakeroot -u* from the given command lines.

To run the test suite use the following command:

```
fakeroot -u tox  # run all tests
```

Some more advanced examples:

```
# verify a changed tox.ini (run this after any change to tox.ini):
fakeroot -u tox --recreate

fakeroot -u tox -e py34  # run all tests, but only on python 3.4

fakeroot -u tox borg.testsuite.locking  # only run 1 test module

fakeroot -u tox borg.testsuite.locking -- -k '"not Timer"'  # exclude some tests

fakeroot -u tox borg.testsuite -- -v  # verbose py.test
```

Important notes:

- When using `--` to give options to py.test, you MUST also give `borg.testsuite[.module]`.

Regenerate usage files

Usage and API documentation is currently committed directly to git, although those files are generated automatically from the source tree.

When a new module is added, the `docs/api.rst` file needs to be regenerated:

```
./setup.py build_api
```

When a command is added, a commandline flag changed, added or removed, the usage docs need to be rebuilt as well:

```
./setup.py build_usage
```

Building the docs with Sphinx

The documentation (in reStructuredText format, .rst) is in docs/.

To build the html version of it, you need to have sphinx installed:

```
pip3 install sphinx  # important: this will install sphinx with Python 3
```

Now run:

```
cd docs/
make html
```

Then point a web browser at docs/_build/html/index.html.

The website is updated automatically through Github web hooks on the main repository.

Using Vagrant

We use Vagrant for the automated creation of testing environments and borgbackup standalone binaries for various platforms.

For better security, there is no automatic sync in the VM to host direction. The plugin *vagrant-scp* is useful to copy stuff from the VMs to the host.

Usage:

```
# To create and provision the VM:
vagrant up OS
# To create an ssh session to the VM:
vagrant ssh OS command
# To shut down the VM:
vagrant halt OS
# To shut down and destroy the VM:
vagrant destroy OS
# To copy files from the VM (in this case, the generated binary):
vagrant scp OS:/vagrant/borg/borg.exe .
```

Creating standalone binaries

Make sure you have everything built and installed (including llfuse and fuse). When using the Vagrant VMs, pyinstaller will already be installed.

With virtual env activated:

```
pip install pyinstaller  # or git checkout master
pyinstaller -F -n borg-PLATFORM borg/__main__.py
for file in dist/borg-*; do gpg --armor --detach-sign $file; done
```

If you encounter issues, see also our *Vagrantfile* for details.

Note: Standalone binaries built with pyinstaller are supposed to work on same OS, same architecture (x86 32bit, amd64 64bit) without external dependencies.

Creating a new release

Checklist:

- make sure all issues for this milestone are closed or moved to the next milestone
- find and fix any low hanging fruit left on the issue tracker
- check that Travis CI is happy
- update `CHANGES.rst`, based on `git log $PREVIOUS_RELEASE..`
- check version number of upcoming release in `CHANGES.rst`
- verify that `MANIFEST.in` and `setup.py` are complete
- `python setup.py build_api ; python setup.py build_usage` and commit
- tag the release:

```
git tag -s -m "tagged/signed release X.Y.Z" X.Y.Z
```

- create a clean repo and use it for the following steps:

```
git clone borg borg-clean
```

  This makes sure no uncommitted files get into the release archive. It also will find if you forgot to commit something that is needed. It also makes sure the vagrant machines only get committed files and do a fresh start based on that.

- run tox and/or binary builds on all supported platforms via vagrant, check for test failures
- create a release on PyPi:

```
python setup.py register sdist upload --identity="Thomas Waldmann" --sign
```

- close release milestone on Github
- announce on:
- Mailing list
- Twitter (follow @ThomasJWaldmann for these tweets)
- IRC channel (change `/topic`)
- create a Github release, include:
  - standalone binaries (see above for how to create them)
    - for OS X, document the OS X Fuse version in the README of the binaries. OS X FUSE uses a kernel extension that needs to be compatible with the code contained in the binary.
  - a link to `CHANGES.rst`

Borg Contributors ("The Borg Collective")

- Thomas Waldmann <tw@waldmann-edv.de>
- Antoine Beaupré <anarcat@debian.org>
- Radek Podgorny <radek@podgorny.cz>
- Yuri D'Elia
- Michael Hanselmann <public@hansmi.ch>
- Teemu Toivanen <public@profnetti.fi>

Borg is a fork of Attic.

Attic authors

Attic is written and maintained by Jonas Borgström and various contributors:

Attic Development Lead

- Jonas Borgström <jonas@borgstrom.se>

Attic Patches and Suggestions

- Brian Johnson
- Cyril Roussillon
- Dan Christensen
- Jeremy Maitin-Shepard
- Johann Klähn
- Petros Moisiadis

- Thomas Waldmann

License

```
Copyright (C) 2015-2016 The Borg Collective (see AUTHORS file)
Copyright (C) 2010-2014 Jonas Borgström <jonas@borgstrom.se>
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.
 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in
    the documentation and/or other materials provided with the
    distribution.
 3. The name of the author may not be used to endorse or promote
    products derived from this software without specific prior
    written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS
OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```