gitblobts Documentation

gitblobts

Apr 19, 2019

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 gitblobts

`gitblobts` is an experimental Python package for **git-backed time-indexed blob storage**. Even so, a lock-in of the stored files with git is avoided. If encryption is not enabled, a lock-in of the file contents with this application is also avoided.

Its goal is to ensure availability of data both locally and remotely. It stores each blob as a file in a preexisting local and remote git repository. Each filename contains an encoded nanosecond timestamp and format version number.

Given the pull and push actions of git, collaborative use of the same remote repo is supported. To prevent merge conflicts, there is a one-to-many mapping of timestamp to filenames. This is accomplished by including sufficient random bytes in the filename to ensure uniqueness.

Subsequent retrieval of blobs is by a time range. At this time there is no implemented method to remove or overwrite a blob; this is by design. From the perspective of the package, once a blob is written, it is considered read-only. An attempt to add a blob with the same timestamp as a preexisting blob will result in a new blob.

An effort has been made to keep third-party package requirements to a minimum.

1.1.1 Links

- Code: https://github.com/impredicative/gitblobts/
- Docs: https://gitblobts.readthedocs.io/
- Release: https://pypi.org/project/gitblobts/

1.1.2 Installation

Using Python 3.7+, install the package from PyPI: `pip install -U gitblobts`.

1.1.3 Usage examples

Storage

```
from typing import Optional
import datetime, gitblobts, json, time, urllib.request

optional_compression_module_name: Optional[str] = [None, 'bz2', 'gzip', 'lzma'][2]
optional_user_saved_encryption_key: Optional[bytes] = [None, gitblobts.generate_
↪key()][1]
```

(continues on next page)

(continued from previous page)

```
store = gitblobts.Store('/path_to/preexisting_git_repo',
                        compression=optional_compression_module_name, key=optional_
↪user_saved_encryption_key)

store.addblob('a byte encoded string'.encode())
store.addblob(b'some bytes' * 1000, timestamp=time.time())
store.addblob(blob=json.dumps([0, 1., 2.2, 3]).encode(),
              timestamp=datetime.datetime.now(datetime.timezone.utc).timestamp())
store.addblob(blob=urllib.request.urlopen('https://i.imgur.com/3GmPd7O.png').read())

store.addblobs(blobs=[b'first blob', b'another blob'])
store.addblobs(blobs=[b'A', b'B'], timestamps=[time.time(), time.time()])
```

Retrieval

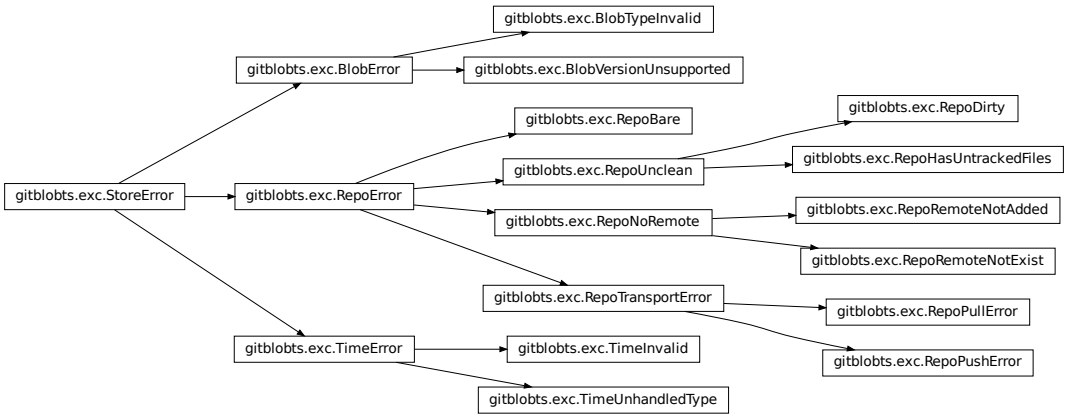
```
from typing import List
from gitblobts import Blob, Store
import time

store = Store('/path_to/preexisting_git_repo', compression='gzip', key=b
↪'JVGmuw3wRntCc7dcQHJ5q1noUs62ydR0Nw8HpyllKn8=')

blobs: List[Blob] = list(store.getblobs(pull=False))
blobs_bytes: List[bytes] = [b.blob for b in blobs]
timestamps: List[float] = [b.timestamp for b in blobs]

blobs2_ascending: List[Blob] = list(store.getblobs(start_time='midnight yesterday',␣
↪end_time='now'))
blobs2_descending: List[Blob] = list(store.getblobs(start_time='now', end_time=
↪'midnight yesterday', pull=True))
blobs3_ascending: List[Blob] = list(store.getblobs(start_time=time.time() - 86400,␣
↪end_time=time.time(), pull=True))
blobs3_descending: List[Blob] = list(store.getblobs(start_time=time.time(), end_
↪time=time.time() - 86400))
```

CHAPTER

TWO

API

2.1 exc



Exceptions.

exception gitblobts.exc.**BlobError**(*msg: str*)
Bases: *gitblobts.exc.StoreError*

exception gitblobts.exc.**BlobTypeInvalid**(*msg: str*)
Bases: *gitblobts.exc.BlobError*

exception gitblobts.exc.**BlobVersionUnsupported**(*msg: str*)
Bases: *gitblobts.exc.BlobError*

exception gitblobts.exc.**RepoBare**(*msg: str*)
Bases: *gitblobts.exc.RepoError*

exception gitblobts.exc.**RepoDirty**(*msg: str*)
Bases: *gitblobts.exc.RepoUnclean*

exception gitblobts.exc.**RepoError**(*msg: str*)
Bases: *gitblobts.exc.StoreError*

exception gitblobts.exc.**RepoHasUntrackedFiles**(*msg: str*)
Bases: *gitblobts.exc.RepoUnclean*

exception gitblobts.exc.**RepoNoRemote**(*msg: str*)
Bases: *gitblobts.exc.RepoError*

exception gitblobts.exc.**RepoPullError**(*msg: str*)
Bases: *gitblobts.exc.RepoTransportError*

exception gitblobts.exc.**RepoPushError**(*msg: str*)
Bases: *gitblobts.exc.RepoTransportError*

exception gitblobts.exc.**RepoRemoteNotAdded**(*msg: str*)
Bases: *gitblobts.exc.RepoNoRemote*

exception gitblobts.exc.**RepoRemoteNotExist**(*msg: str*)
Bases: *gitblobts.exc.RepoNoRemote*

exception gitblobts.exc.**RepoTransportError**(*msg: str*)
Bases: *gitblobts.exc.RepoError*

exception gitblobts.exc.**RepoUnclean**(*msg: str*)
Bases: *gitblobts.exc.RepoError*

exception gitblobts.exc.**StoreError**(*msg: str*)
Bases: Exception

This is the base exception class in this module.

This exception is not raised directly. All other exception classes in this module hierarchically derive from it.

Parameters **msg** – exception error message.

exception gitblobts.exc.**TimeError**(*msg: str*)
Bases: *gitblobts.exc.StoreError*

exception gitblobts.exc.**TimeInvalid**(*msg: str*)
Bases: *gitblobts.exc.TimeError*

exception gitblobts.exc.**TimeUnhandledType**(*msg: str*)
Bases: *gitblobts.exc.TimeError*

2.2 store

class gitblobts.store.**Blob**(*timestamp: float*, *blob: bytes*)
Bases: object

Instances of this class are returned by *Store.getblobs()*.

This class is not meant to be initialized otherwise.

Parameters

- **timestamp** – registered timestamp
- **blob** – content

class gitblobts.store.**Store**(*path: Union[str, pathlib.Path], *, compression: Optional[str] = None, key: Optional[bytes] = None*)
Bases: object

Initialize the interface to a preexisting cloned git repository.

Parameters

- **path** – path to a preexisting cloned git repository. It must have a valid remote.

- **compression** – name of a built-in or third-party importable module with *compress* and *decompress* functions, e.g. `bz2`, `gzip`, `lzma`. Once established, this must not be changed for a given repository, failing which file corruption can result.
- **key** – optional encryption and decryption key as previously generated by `generate_key()`. Once established, this must not be changed for a given repository, failing which file corruption can result. The key should be stored safely. If it is lost, it will not be possible to decrypt previously encrypted blobs. If anyone else gains access to it, it can be used to decrypt blobs.

addblob(*blob: bytes*, *timestamp: Union[None*, *int*, *float*, *str*, *time.struct_time] = None*) → None

Add a blob and also push it to the remote repository.

Parameters

- **blob** – bytes representation of text or an image or anything else.
- **timestamp** – optional time at which to index the blob, preferably as a Unix timestamp. If a Unix timestamp, it can be positive or negative number of whole or fractional seconds since epoch. This doesn't have to be unique, and so there can be a one-to-many mapping of timestamp to blobs. If a string, it is parsed using dateparser.parse. If not specified, the current time is used.

Idempotency, if required, is to be implemented externally.

addblobs(*blobs: Iterable[bytes]*, *timestamps: Optional[Iterable[Union[None, int, float, str, time.struct_time]]] = None*) → None

Add multiple blobs and also push them to the remote repository.

For adding multiple blobs, this method is more efficient than multiple calls to `addblob()`, as the commit and push are batched and done just once.

Parameters

- **blobs** – iterable or sequence.
- **timestamps** – optional iterable or sequence of the same length as *blobs*. If not specified, the current time is used, and this will naturally increment just slightly for each subsequent blob. For further details, refer to the *timestamp* parameter of `addblob()`.

In case the length of *blobs* and *timestamps* are somehow not identical, the shorter of the two lengths is used.

Idempotency, if required, is to be implemented externally.

getblobs(*start_time: Union[None*, *int*, *float*, *str*, *time.struct_time] = -inf*, *end_time: Union[None*, *int*, *float*, *str*, *time.struct_time] = inf*, ***, *pull: Optional[bool] = False*) → Iterator[gitblobts.store.Blob]

Yield blobs matching the specified time range.

This method currently requires listing and decoding the metadata for all files in the repository directory. From this perspective, calls to it should be consolidated.

Parameters

- **start_time** – inclusive start time. Refer to the corresponding type annotation, and also to the *timestamp* parameter of `addblob()`.
- **end_time** – inclusive end time. Refer to the corresponding type annotation, and also to the *timestamp* parameter of `addblob()`.
- **pull** – pull first from remote repository. A pull should be avoided unless necessary.

Yields instances of *Blob*. If *start_time* ≤ *end_time*, blobs are yielded in ascending chronological order sorted by their registered timestamp, otherwise in descending order.

To pull without yielding any blobs, one can therefore call `get_blobs(math.inf, math.inf, pull=True)`.

gitblobts.store.**generate_key**() → bytes

Return a random new Fernet key.

The key should be stored safely. If it is lost, it will not be possible to decrypt previously encrypted blobs. If anyone else gains access to it, it can be used to decrypt blobs.

An example of a generated key is `b'NrYgSuzXVRWtarWcczyuwFs6vZftN1rnlzZtGDaV7iE='`.

Returns key used for encryption and decryption.

CHAPTER

THREE

INDEX

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX