# passpy Documentation

***Release 1.0.2***

**Benedikt Rascher-Friesenhausen**

**Feb 22, 2020**

# Contents

passpy has been written to be a platform independent library and cli that is compatible with ZX2C4's pass.

passpy saves your passwords in gpg encrypted files and optionally uses git as a revision tool. All files are stored inside the directory given by the *PASSWORD_STORE_DIR* environment variable (*~/.password-store* if not set) and can be organised into folders. You can also just copy the whole store to have your passwords available where ever you like.

CHAPTER 1

# Installation

## 1.1 PyPI

Just do `$ [sudo] pip install passpy`

## 1.2 Arch Linux

The package `python-passpy` is available in the AUR for you to install however you like.

## 1.3 Manually

Either clone the git repository using `$ git clone https://github.com/bfrascher/passpy.git`

or download the source from the releases tab and extract it. Afterwards change into the new folder and do `$ [sudo] python setup.py install`

CHAPTER 2

# Dependencies

passpy depends on Python 3.3 or later (it has mostly been tested using Python 3.5). The program makes use of git and gpg2 as well as either xclip or xsel on Linux.

The following Python packages will be installed alongside passpy:

- gitpython
- python-gnupg
- click
- pyperclip

If you are on Windows and want colourised output on the command line, you will additionally need to install colorama.

CHAPTER 3

# Changelog

## 3.1 1.0.2

- Now also read the default password store location from *PASSWORD_STORE_DIR* for the cli.
- Show a better error message when the password store does not exist.
- Always use the *path* parameter of *Store.init_store* relative to *Store.store_dir*.

## 3.2 1.0.1

- Fix documentation.

## 3.3 1.0.0

- Read the default password store location from the *PASSWORD_STORE_DIR* environment variable, just like *pass* does (contributed by Jonathan Waldrep).

CHAPTER 4

Contents

## 4.1 Usage

### 4.1.1 CLI

#### Setting up the password store

To initialise a new password store use:

```
$ passpy init "passpy gpg id"
Password store initialised for passpy gpg id.
```

where `passpy gpg id` is the ID of the GPG key to encrypt the password files with. You can use different IDs for different folders inside the store by adding the `-path` or `-p` option. It is also possible to use multiple IDs instead of just one.

If you want to use git to revision your passwords you can initialise it with:

```
$ passpy git init
```

By calling `passpy git [...]` you can directly interact with git acting on the password store to e.g. add remotes to push/pull to/from them.

#### Using the password store

You can use the `--help` option on any command to get all the available options.

To list all existing passwords in the password store use:

```
$ passpy ls
Password Store
|-- Email
```

(continues on next page)

(continued from previous page)

```
|   |-- google.com
|   `-- yahoo.com
|-- Programming
|   |-- github.com
|   `-- Python
|       |-- python.org
|       `-- readthedocs.org
`-- Notes
    `-- Wi-Fi
        |-- home
        `-- work
```

We can show a password:

```
$ passpy show Email/google.com
z.Rw6$`U=2MZs(i9\>-r
```

or copy it to the clipboard:

```
$ passpy show -c Email/google.com
Copied Email/google.com to the clipboard.
```

When accessing a password you will be prompted to enter your password for the encryption key. If you have a running `gpg-agent` you can configure it, so that you stay authenticated for several minutes. This helps especially when accessing multiple passwords in short order, e.g. when moving passwords and reencrypting them.

To add an existing password to the store use:

```
$ passpy insert Webshop/amazon.com
Enter password for Webshop/amazon.com:
Repeat for confirmation:
```

Using the `--echo` or `-e` option you won't be prompted to repeat the password. With `--multiline` or `-m` you can enter multiple lines, or you can use `$ passpy edit pass-name` to edit password files with your default text editor.

To let passpy generate a password for you, use:

```
$ passpy generate Social/facebook.com 16
The generated password for Social/facebook.com is:
&,"S_Bq}qWKW&<^f
```

If you don't want any symbols in your password use the `--no-symbols` or `-n` option. Like `show` you can copy the generated password to the clipboard with `--clip` or `-c` and `--in-place` or `-i` will overwrite the first line of an existing password file with the new password.

To copy or move a password file or folder in the password store use:

```
$ passpy cp/mv Webshop Webshops
/home/user/.password-store/Webshop/amazon.com.gpg -> /home/user/.password-store/
↪Webshops/amazon.com.gpg
```

To avoid being prompted for every file that already exists at the destination, use the `--force` or `-f` option. When using a trailing `/` in the destination name, the destination will always be treated as a directory.

Finally, you can delete a password file

```
$ passpy rm Social/facebook.com
Really delete Social/facebook.com? [y/N] y
removed Social/facebook.com
```

Passing the `--force` or `-f` option will delete the file without asking and `--recursive` or `-r` will delete whole directories, if one is given.

### 4.1.2 Library

To use passpy in your Python project, we will first have to create a new *passpy.store.Store* object

```
>>> import passpy
>>> store = passpy.Store()
```

If git or gpg2 are not in your PATH you will have to specify them via `git_bin` and `gpg_bin` when creating the `store` object. You can also create the store on a different folder, be passing `store_dir` along.

To initialise the password store at `store_dir`, if it isn't already, use

```
>>> store.init_store('store gpg id')
```

where `store gpg id` is the name of a GPG ID. Optionally, git can be initialised in very much the same way

```
>>> store.init_git()
```

You are now ready to interact with the password store. You can set and get keys using *passpy.store.Store.set_key()* and *passpy.store.Store.get_key()*. *passpy.store.Store.gen_key()* generates a new password for a new or existing key. To delete a key or directory, use *passpy.store.Store.remove_path()*.

For a full overview over all available methods see *store module*.

### 4.1.3 Data Organisation

You are free to organise your files in the store however you like. But, as the `--clip` or `-c` option only copies the first line of a password file to the clipboard and the `--in-place` or `-i` option overwrites the first line with a new password, it is recommended that you have your password on the first line for each password file. That way it is easy to fetch a password for a login form or update an existing password file.

Some users might want to store additional information for a store entry, like a websites URL, the username and so on. There are many methods to do this, some of which are listed under *Data Organization* on the website for ZX2C4's pass. The authors preferred way to do this (both for pass and passpy) is to have additional lines under the first one with a leading keyword. An entry might look like this:

```
z.Rw6$`U=2MZs(i9\>-r
URL: accounts.google.com/*
Username: somegoogleuser@gmail.com

Chrome Sync Password: EK6zzRo4chejRBztuVUF3CvqvRg9E4
```

Of course, as said in the beginning of the section, how you organise your data is completely up to you and this is just one way of doing things.

## 4.2 Differences to ZX2C4's pass

While passpy is fully compatible with [ZX2C4's pass](http://www.passwordstore.org), there are some differences:

- As for the moment passpy does not print as many messages to the user, as pass does. Also some messages might be phrased differently.
- When invoking `pass` without any known command, `show` is used. passpy always needs to be given a command to invoke.
- passpy allows `gen` as a alternative to `generate`.
- After editing a password file with an editor pass notes the used editor in the git commit message. passpy uses the same message for updating a key, regardless if an editor was used or not. This is because for both cases *passpy.store.Store.set_key()* is being called which also handles the git commit.
- pass lists all files in the password store that do not start with a `.`. passpy only lists files that end on `.gpg`. The reason for this change is that the returned key names should be directly accessible with *passpy.store.Store.get_key()*, which expects a file ending on `.gpg`.

## 4.3 API Reference

### 4.3.1 git module

This module includes all calls to the git wrapper.

passpy.git.**_git_commit**(*repo*, *msg*, *verbose=False*)
Commit the current changes.

**Parameters**

- **repo** (*git.repo.base.Repo*) – The repository to use.
- **msg** (*str*) – The commit message.
- **verbose** (*bool*) – (optional) If `True` git's standard output will be printed.

passpy.git.**get_git_repository**(*path*)
Get the git repository at path.

**Parameters** **path** (*str*) – The path of a git repository to return.

**Return type** *git.repo.base.Repo*

**Returns** The git repository at path or None if no repository exists.

passpy.git.**git_add_path**(*repo*, *path*, *msg*, *commit=True*, *verbose=False*)
Add a file or directory to the git repository and commit.

**Parameters**

- **repo** (*git.repo.base.Repo*) – The git repository. If `None` the function will silently fail.
- **path** (*str or list*) – The path of the file or directory to commit relative to `passpy.store.Store.store_dir`.
- **msg** (*str*) – The commit message.
- **commit** (*bool*) – (optional) If `True` the added file will also be commited.
- **verbose** (*bool*) – (optional) If `True` git's standard output will be printed.

**Raises** **OSError** – if something went wrong with adding the files.

passpy.git.**git_config**(*repo*, **args*)
Change the configuration of a git repository.

**Parameters** **repo** (git.Repo) – The git repository to change the configuration for.

passpy.git.**git_init**(*path*)
Create a new git repository.

**Parameters** **path** (*str*) – The absolute path directory to create a git repository in.

**Return type** git.Repo

**Returns** The newly initialised git repository.

passpy.git.**git_remove_path**(*repo*, *path*, *msg*, *recursive=False*, *commit=True*, *verbose=False*)
Remove the file or directory at path from the repository and commit.

**Parameters**

- **repo** (git.repo.base.Repo) – The git repository. If None the function will silently fail.
- **path** (*str or list*) – The file or directory to remove.
- **msg** (*str*) – The commit message.
- **recursive** (*bool*) – (optional) Set to True if directories should be removed from the repository recursively.
- **verbose** (*bool*) – (optional) If True git's standard output will be printed.

### 4.3.2 gpg module

This module includes all calls to the gnupg wrapper.

passpy.gpg.**_get_gpg_recipients**(*path*)
Get the GPG recipients for the given path.

**Parameters** **path** (*str*) – The directory to get the GPG recipients for.

**Raises** **FileNotFoundError** – if there is not valid .gpg-id file for path.

**Return type** list

**Returns** The list of IDs of the GPG recipients for the given path.

passpy.gpg.**_reencrypt_key**(*path*, *gpg*, *gpg_recipients*)
Reencrypt a single key.

Gets called from passpy.gpg._reencrypt_path().

**Parameters**

- **path** (*str*) – The path to a gpg encrypted file.
- **gpg** (gnupg.GPG) – The gpg object.
- **gpg_recipients** (*list*) – The list of GPG Ids to encrypt the key with.

passpy.gpg.**read_key**(*path*, *gpg_bin*, *gpg_opts*)
Read and decrypt a single key file.

**Parameters**

- **path** (*str*) – The path to the key to decrypt.

- **gpg_bin** (*str*) – The path to the gpg binary.
- **gpg_opts** (*list*) – The options for gpg.

**Return type** str

**Returns** The unencrypted content of the file at *path*.

passpy.gpg.**reencrypt_path**(*path*, *gpg_bin*, *gpg_opts*)
Reencrypt a single or multiple keys.

If path is a directory all keys inside that directory and it's subdirectories will be reencrypted.

**Parameters**

- **path** (*str*) – The key or directory to reencrypt. If `None` the function will silently fail.
- **gpg_bin** (*str*) – The path to the gpg binary.
- **gpg_opts** (*list*) – The gpg options.

**Raises** **FileNotFoundError** – if path does not exist.

passpy.gpg.**write_key**(*path*, *key_data*, *gpg_bin*, *gpg_opts*)
Encrypt and write a single key file.

**Parameters**

- **path** (*str*) – The path to the key to decrypt.
- **gpg_bin** (*str*) – The path to the gpg binary.
- **gpg_opts** (*list*) – The options for gpg.

### 4.3.3 store module

**class** passpy.store.**Store**(*gpg_bin='gpg2'*, *git_bin='git'*, *store_dir='~/.password-store'*, *use_agent=True*, *interactive=False*, *verbose=False*)
Python implementation of ZX2C4's password store.

**__init__**(*gpg_bin='gpg2'*, *git_bin='git'*, *store_dir='~/.password-store'*, *use_agent=True*, *interactive=False*, *verbose=False*)
Creates a new Store object.

**Parameters**

- **gpg_bin** (*str*) – (optional) The path to the gpg binary.
- **git_bin** (*str*) – (optional) The path to the git binary. CURRENTLY DOES NOTHING You will need to set the environmental variable GIT_PYTHON_GIT_EXECUTABLE to your path to git binary if your git binary not in your PATH already.
- **store_dir** (*str*) – (optional) The path to the password store. Will use the value of the PASSWORD_STORE_DIR environment variable by default, or *~/.password-store*, if not set.
- **use_agent** (*bool*) – (optional) Set to `True` if you are using a gpg agent.
- **interactive** (*bool*) – (optional) If `True` the user will be prompted before overwriting/deleting files.
- **verbose** (*bool*) – (optional) If `True` additional information will be printed to the standard out.

**__weakref__**
list of weak references to the object (if defined)

**_copy_move_path**(*old_path*, *new_path*, *force=False*, *move=False*)
Copies or moves a key or directory within the password store.

**Parameters**

- **old_path** (*str*) – The current path of the key or directory.
- **new_path** (*str*) – The new path of the key or directory. If *new_path* ends in a trailing '/' it will always be treated as a directory.
- **force** (*bool*) – If `True` any existing key or directory at *new_path* will be overwritten.
- **move** (*bool*) – If `True` the key or directory will be moved. If `False` the key or directory will be copied instead.

**_get_store_name**(*path*)
Returns the path relative to the store.

**Parameters** **path** (*str*) – The absolute path to an entry in the store.

**Return type** str

**Returns** *path* relative to `passpy.store.Store.store_dir` without a leading '/' and trailing '.gpg' if any.

**copy_path**(*old_path*, *new_path*, *force=False*)
Copies a key or directory within the password store.

**Parameters**

- **old_path** (*str*) – The current path of the key or directory.
- **new_path** (*str*) – The new path of the key or directory. If *new_path* ends in a trailing '/' it will always be treated as a directory.
- **force** (*bool*) – If `True` any existing key or directory at *new_path* will be overwritten.

**find**(*names*)
Find keys by name.

Finds any keys in the password store that contain any one entry in *names*.

**Parameters** **names** (*str or list*) – The name or names to find keys for.

**Return type** list

**Returns** A list of keys whose name contain any one entry in *names*.

**gen_key**(*path*, *length*, *symbols=True*, *force=False*, *inplace=False*)
Generate a new password for a key.

**Parameters**

- **path** (*str*) – The path of the key.
- **length** (*int*) – The length of the new password.
- **symbols** (*bool*) – (optional) If `True` non alphanumeric characters will also be used in the new password.
- **force** (*bool*) – (optional) If `True` an existing key at *path* will be overwritten.
- **inplace** (*bool*) – (optional) If `True` only the first line of an existing key at *path* will be overwritten with the new password.

**get_key**(*path*)
Reads the data of the key at path.

**Parameters** **path** (*str*) – The path to the key (without '.gpg' ending) relative to `passpy.store.Store.store_dir`.

**Return type** str

**Returns** The key data as a string or `None`, if the key does not exist.

**Raises** **FileNotFoundError** – if *path* is not a file.

**init_git**()
Initialise git for the password store.

Silently fails if `passpy.store.Store.repo` is not `None`.

**init_store**(*gpg_ids*, *path=None*)
Initialise the password store or a subdirectory with the gpg ids.

**Parameters**

- **gpg_ids** (*list*) – The list of gpg ids to encrypt the password store with. If the list is empty, the current gpg id will be removed from the directory in path or root, if path is None.
- **path** (*str*) – (optional) If given, the gpg ids will only be set for the given directory. The path is relative to `passpy.store.Store.store_dir`.

**Raises**

- **ValueError** – if the there is a problem with *path*.
- **FileExistsError** – if `passpy.store.Store.store_dir` already exists and is a file.
- **FileNotFoundError** – if the current gpg id should be deleted, but none exists.
- **OSError** – if the directories in path do not exist and can't be created.

**list_dir**(*path*)
Returns all directory and key entries for the given path.

**Parameters** **path** (*str*) – The directory to list relative to `passpy.store.Store.store_dir`

**Return type** (list, list)

**Returns** Two lists, the first for directories, the second for keys. `None` if *path* is not a directory.

**Raises** **FileNotFoundError** – if *path* is not a directory in the password store.

**move_path**(*old_path*, *new_path*, *force=False*)
Moves a key or directory within the password store.

**Parameters**

- **old_path** (*str*) – The current path of the key or directory.
- **new_path** (*str*) – The new path of the key or directory. If *new_path* ends in a trailing '/' it will always be treated as a directory.
- **force** (*bool*) – If `True` any existing key or directory at *new_path* will be overwritten.

**remove_path**(*path*, *recursive=False*, *force=False*)
Removes the given key or directory from the store.

**Parameters**

- **path** (*str*) – The key or directory to remove. Use '' to delete the whole store.
- **recursive** (*bool*) – (optional) Set to `True` if nonempty directories should be removed.
- **force** (*bool*) – (optional) If `True` the user will never be prompted for deleting a file or directory, even if `passpy.store.Store.interactive` is set.

**search**(*term*)
Search through all keys.

**Parameters** **term** (*str*) – The term to search for. The term will be compiled as a regular expression.

**Return type** dict

**Returns** The dictionary has an entry for each key, that matched the given term. The entry for that key then contains a list of tuples with the line the term was found on and the match object.

**set_key**(*path*, *key_data*, *force=False*)
Add a key to the store or update an existing one.

**Parameters**

- **path** (*str*) – The key to write.
- **key_data** (*str*) – The data of the key.
- **foce** (*bool*) – (optional) If `True` path will be overwritten if it exists.

**Raises** **FileExistsError** – if a key already exists for path and overwrite is `False`.

### 4.3.4 util module

passpy.util.**copy_move**(*src*, *dst*, *force=False*, *move=False*, *interactive=False*, *verbose=False*)
Copies/moves a file or directory recursively.

This function is partially based on the *cp* function from the pycoreutils package written by Hans van Leeuwen and licensed under the MIT license.

**Parameters**

- **src** (*str*) – The file or directory to be copied.
- **dst** (*str*) – The file or directory to be copied to.
- **force** (*bool*) – If `True` existing files at the destination will be silently overwritten.
- **interactive** (*bool*) – If `True` the user will be prompted for every file to be overwritten. Has no effect if *force* is also `True`.
- **verbose** (*bool*) – If `True` print the old and new filename for every copied/moved file.

**Raises**

- **FileNotFoundError** – if there exists no key or directory for *src*.
- **FileExistsError** – if a key at *dst* already exists and *force* is set to `False`.

passpy.util.**gen_password**(*length*, *symbols=True*)
Generates a random string.

Uses `random.SystemRandom` if available and `random.Random` otherwise.

**Parameters**

- **length** (*int*) – The length of the random string.
- **symbols** (*bool*) – (optional) If `True` `string.punctuation` will also be used to generate the output.

**Return type** str

**Returns** A random string of length *length*.

passpy.util.**initialised**(*func*)

Check that the store is initialised before running.

Used as a decorator in methods for *passpy.store.Store*.

**Parameters** **func** – A method of *passpy.store.Store*.

**Return type** function

**Returns** The method if the store is initialised.

**Raises** **passpy.exceptions.StoreNotInitialisedError** – if the store is not initialised.

passpy.util.**trap**(*path_index*)

Prevent accessing files and directories outside the password store.

*path_index* is necessary as the functions that need to be trapped have different argument lists. This way we can indicate which argument contains the paths that are to be checked.

**Parameters** **path_index** (*int or str*) – The index for the path variable in either *args* or *kwargs*.

**Return type** func

**Returns** The trapped function.

CHAPTER 5

# Indices and tables

- genindex
- search

# Python Module Index

## p

# Index