qpass

Release 2.3

Dec 03, 2018

Contents

Welcome to the documentation of *qpass* version 2.3! The following sections are available:

CHAPTER 1

User documentation

The readme is the best place to start reading, it's targeted at all users and documents the command line interface:

1.1 qpass: Frontend for pass (the standard unix password manager)

The qpass program is a simple command line frontend for pass, the standard unix password manager. It makes it very easy to quickly find and copy specific passwords in your `~/.password-store` to the clipboard. The package is currently tested on cPython 2.6, 2.7, 3.4, 3.5, 3.6 and PyPy (2.7). It's intended to work on Linux as well as macOS, although it has only been tested on Linux.

1.1.1 Installation

The qpass package is available on PyPI which means installation should be as simple as:

```
$ pip install qpass
```

There's actually a multitude of ways to install Python packages (e.g. the per user site-packages directory, virtual environments or just installing system wide) and I have no intention of getting into that discussion here, so if this intimidates you then read up on your options before returning to these instructions ;-).

1.1.2 Usage

There are two ways to use the qpass package: As the command line program `qpass` and as a Python API. For details about the Python API please refer to the API documentation available on Read the Docs. The command line interface is described below.

Command line

Usage: *qpass [OPTIONS] KEYWORD..*

Search your password store for the given keywords or patterns and copy the password of the matching entry to the clipboard. When more than one entry matches you will be prompted to select the password to copy.

If you provide more than one KEYWORD all of the given keywords must match, in other words you're performing an AND search instead of an OR search.

Instead of matching on keywords you can also enter just a few of the characters in the name of a password, as long as those characters are in the right order. Some examples to make this more concrete:

- The pattern 'pe/zbx' will match the name 'Personal/Zabbix'.
- The pattern 'ba/cc' will match the name 'Bank accounts/Creditcard'.

When a password is copied to the clipboard, any text after the first line will be shown on the terminal, to share any additional details about the password entry (for example the associated username or email address). The `-q`, `--quiet` option suppresses this text.

Supported options:

Option	Description
`-e`, `--edit`	Edit the matching entry instead of copying it to the clipboard.
`-l`, `--list`	List the matching entries on standard output.
`-n`, `--no-clipboard`	Don't copy the password of the matching entry to the clipboard, instead show the password on the terminal (by default the password is copied to the clipboard but not shown on the terminal).
`-p`, `--password-store=DIRECTORY`	Search the password store in `DIRECTORY`. If this option isn't given the password store is located using the `$PASSWORD_STORE_DIR` environment variable. If that environment variable isn't set the directory ~/.password-store is used. You can use the `-p`, `--password-store` option multiple times to search more than one password store at the same time. No distinction is made between passwords in different password stores, so the names of passwords need to be recognizable and unique.
`-f`, `--filter=PATTERN`	Don't show lines in the additional details which match the case insensitive regular expression given by `PATTERN`. This can be used to avoid revealing sensitive details on the terminal. You can use this option more than once.
`-x`, `--exclude=GLOB`	Ignore passwords whose name matches the given `GLOB` filename pattern. This argument can be repeated to add multiple exclude patterns.
`-v`, `--verbose`	Increase logging verbosity (can be repeated).
`-q`, `--quiet`	Decrease logging verbosity (can be repeated).
`-h`, `--help`	Show this message and exit.

1.1.3 Why use pass?

In 2016 I was looking for a way to securely share passwords and other secrets between my laptops and smartphones. I'm not going to bore you with the full details of my quest to find the ultimate password manager but I can highlight a few points about pass that are important to me:

- *GPG encryption*
- *Git version control*
- *SSH secure transport*

GPG encryption

GPG is a cornerstone of computer security and it's open source. This means it receives quite a lot of peer review, which makes it easier for me to trust (versus do-it-yourself cryptography). Because pass uses GPG to implement its encryption my trust extends directly to pass. Of course it also helps that I had years of experience with GPG before I started using pass :-).

Git version control

The git integration in pass makes it very easy to keep your passwords under version control and synchronize the passwords between multiple systems. Git is a great version control system and while I sometimes get annoyed by the fact that `git pull` automatically merges, it's actually the perfect default choice for a password store. As an added bonus you have a history of every change you ever made to your passwords.

SSH secure transport

I've been using SSH to access remote systems over secure connections for *a very long time* now so I'm quite comfortable setting up and properly securing SSH servers. In the case of pass I use SSH to synchronize my passwords between my laptops and smartphones via a central server that hosts the private git repository.

1.1.4 History

Shortly after starting to use pass I realized that I needed a quick and easy way to copy any given password to the clipboard, something smarter than the pass program.

I tried out several GUI frontends but to be honest each of them felt clumsy, I guess that through my work as a system administrator and programmer I've grown to prefer command line interfaces over graphical user interfaces :-). For a few weeks I tried upass (a somewhat fancy command line interface) but the lack of simple things like case insensitive search made me stop using it.

Out of frustration I hacked together a simple Python script that would perform case insensitive substring searches on my passwords, copying the password to the clipboard when there was exactly one match. I called the Python script qpass, thinking that it was similar in purpose to upass but much quicker for me to use, so *q* (for quick) instead of *u*.

After using that Python script for a while I noticed that case insensitive substring searching still forced me to specify long and detailed patterns in order to get a unique match. Experimenting with other ways to match unique passwords I came up with the idea of performing a "fuzzy match" against the pathname of the password (including the directory components). The fuzzy searching where a pattern like `e/z` matches `Personal/Zabbix` has since become my primary way of interacting with my password stores.

Support for multiple password stores

One great aspect of pass is the git integration that makes it easy to share a password store between several devices[1] or people[2]. This use case makes it much more likely that you'll end up using multiple password stores, which is something that pass doesn't specifically make easy.

This is why I added support for querying multiple password stores to qpass in version 2.0. For now I've kept things simple which means no distinction is made between passwords in different password stores, so the names of passwords need to be recognizable and unique.

About the name

As explained above I initially wrote and named qpass with no intention of ever publishing it. However since then my team at work has started using pass to manage a shared pasword store and ever since we started doing that I've missed the ability to query that password store using qpass :-).

Publishing qpass as an open source project with a proper Python package available on PyPI provides a nice way to share qpass with my team and it also forces me to maintain proper documentation and an automated test suite.

While considering whether to publish qpass I found that there's an existing password manager out there called QPass. I decided not to rename my project for the following reasons:

- While both projects are password managers, they are intended for very different audiences (I'm expecting my end users to be power users that are most likely system administrators and/or programmers).

[1] For example I synchronize my password store between my personal laptop and my work laptop and I also have access to my password store on my smartphones (thanks to the Android application Password Store).

[2] My team at work also uses pass so because I was already using pass for personal use, I now find myself frequently searching through multiple password stores.

- I consider the name of the executable of a GUI program to be a lot less relevant than the name of the executable of a command line program. This is because the GUI will most likely be started via an application launcher, which means the executable doesn't even need to be on the `$PATH`.
- Let's be honest, pass is already for power users only, so my qpass frontend is most likely not going to see a lot of users ;-).

1.1.5 Contact

The latest version of qpass is available on PyPI and GitHub. The documentation is hosted on Read the Docs and includes a changelog. For bug reports please create an issue on GitHub. If you have questions, suggestions, etc. feel free to send me an e-mail at peter@peterodding.com.

1.1.6 License

This software is licensed under the MIT license.

CHAPTER 2

API documentation

The following API documentation is automatically generated from the source code:

2.1 API documentation

The following documentation is based on the source code of version 2.3 of the *qpass* package.

2.1.1 qpass

Frontend for pass, the standard unix password manager.

qpass.**DEFAULT_DIRECTORY = '~/.password-store'**
The default password storage directory (a string).

The value of *DEFAULT_DIRECTORY* is normalized using parse_path().

qpass.**DIRECTORY_VARIABLE = 'PASSWORD_STORE_DIR'**
The environment variable that sets the password storage directory (a string).

class qpass.**AbstractPasswordStore**(**kw)
Abstract Python API to query passwords managed by pass.

This abstract base class has two concrete subclasses:

- The *QuickPass* class manages multiple password stores as one.
- The *PasswordStore* class manages a single password store.

entries
A list of *PasswordEntry* objects.

exclude_list
A list of strings with filename patterns to ignore.

The fnmatch module is used for pattern matching. Filenames as well as patterns are normalized to lowercase before pattern matching is attempted.

Note: The *exclude_list* property is a custom_property. You can change the value of this property using normal attribute assignment syntax. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use del or delattr().

---

filtered_entries
A list of *PasswordEntry* objects that don't match the exclude list.

Note: The *filtered_entries* property is a cached_property. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use del or delattr().

---

fuzzy_search(**filters*)
Perform a "fuzzy" search that matches the given characters in the given order.

Parameters **filters** – The pattern(s) to search for.

Returns The matched password names (a list of strings).

select_entry(**arguments*)
Select a password from the available choices.

Parameters **arguments** – Refer to *smart_search()*.

Returns The name of a password (a string) or None (when no password matched the given *arguments*).

simple_search(**keywords*)
Perform a simple search for case insensitive substring matches.

Parameters **keywords** – The string(s) to search for.

Returns The matched password names (a generator of strings).

Only passwords whose names matches *all* of the given keywords are returned.

smart_search(**arguments*)
Perform a smart search on the given keywords or patterns.

Parameters **arguments** – The keywords or patterns to search for.

Returns The matched password names (a list of strings).

Raises The following exceptions can be raised:

- *NoMatchingPasswordError* when no matching passwords are found.
- *EmptyPasswordStoreError* when the password store is empty.

This method first tries *simple_search()* and if that doesn't produce any matches it will fall back to *fuzzy_search()*. If no matches are found an exception is raised (see above).

class qpass.**QuickPass**(***kw*)
Python API to query multiple password stores as if they are one.

See also The *PasswordStore* class to query a single password store.

repr_properties = ['stores']
The properties included in the output of repr().

entries
A list of *PasswordEntry* objects.

Note: The *entries* property is a cached_property. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use del or delattr().

stores
A list of *PasswordStore* objects.

Note: The *stores* property is a custom_property. You can change the value of this property using normal attribute assignment syntax. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use del or delattr().

class qpass.**PasswordStore**(***kw*)
Python API to query a single password store.

See also The *QuickPass* class to query multiple password stores.

repr_properties = ['directory', 'entries']
The properties included in the output of repr().

context
An execution context created using executor.contexts.

The value of *context* defaults to a LocalContext object with the following characteristics:

- The working directory of the execution context is set to the value of *directory*.
- The environment variable given by *DIRECTORY_VARIABLE* is set to the value of *directory*.

Raises *MissingPasswordStoreError* when *directory* doesn't exist.

Note: The *context* property is a custom_property. You can change the value of this property using normal attribute assignment syntax. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use del or delattr().

directory
The pathname of the password storage directory (a string).

When the environment variable given by *DIRECTORY_VARIABLE* is set the value of that environment variable is used, otherwise *DEFAULT_DIRECTORY* is used. In either case the resulting directory pathname is normalized using parse_path().

When you set the *directory* property, the value you set will be normalized using parse_path() and the computed value of the *context* property is cleared.

Note: The *directory* property is a custom_property. You can change the value of this property using normal attribute assignment syntax. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use del or delattr().

entries
A list of *PasswordEntry* objects.

Note: The *entries* property is a cached_property. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use del or delattr().

ensure_directory_exists()
Make sure *directory* exists.

Raises *MissingPasswordStoreError* when the password storage directory doesn't exist.

class qpass.**PasswordEntry**(***kw*)
PasswordEntry objects bind the name of a password to the store that contains the password.

repr_properties = ['name']
The properties included in the output of repr().

context
The *context* of *store*.

name
The name of the password store entry (a string).

Note: The *name* property is a required_property. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *name* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

password
The password identified by *name* (a string).

Note: The *password* property is a cached_property. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use del or delattr().

store
The *PasswordStore* that contains the entry.

Note: The *store* property is a required_property. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *store* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

text
The full text of the entry (a string).

Note: The *text* property is a cached_property. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use del or delattr().

copy_password()
Copy the password to the clipboard.

format_text(*include_password=True*, *use_colors=None*, *padding=True*, *filters=()*)
Format *text* for viewing on a terminal.

Parameters

- **include_password** – True to include the password in the formatted text, False to exclude the password from the formatted text.
- **use_colors** – True to use ANSI escape sequences, False otherwise. When this is None terminal_supports_colors() will be used to detect whether ANSI escape sequences are supported.
- **padding** – True to add empty lines before and after the entry and indent the entry's text with two spaces, False to skip the padding.
- **filters** – An iterable of regular expression patterns (defaults to an empty tuple). If a line in the entry's text matches one of these patterns it won't be shown on the terminal.

Returns The formatted entry (a string).

qpass.**create_fuzzy_pattern**(*pattern*)
Convert a string into a fuzzy regular expression pattern.

Parameters **pattern** – The input pattern (a string).

Returns A compiled regular expression object.

This function works by adding `.*` between each of the characters in the input pattern and compiling the resulting expression into a case insensitive regular expression.

2.1.2 qpass.cli

Usage: *qpass [OPTIONS] KEYWORD..*

Search your password store for the given keywords or patterns and copy the password of the matching entry to the clipboard. When more than one entry matches you will be prompted to select the password to copy.

If you provide more than one KEYWORD all of the given keywords must match, in other words you're performing an AND search instead of an OR search.

Instead of matching on keywords you can also enter just a few of the characters in the name of a password, as long as those characters are in the right order. Some examples to make this more concrete:

- The pattern 'pe/zbx' will match the name 'Personal/Zabbix'.
- The pattern 'ba/cc' will match the name 'Bank accounts/Creditcard'.

When a password is copied to the clipboard, any text after the first line will be shown on the terminal, to share any additional details about the password entry (for example the associated username or email address). The -q, --quiet option suppresses this text.

Supported options:

Option	Description
-e, --edit	Edit the matching entry instead of copying it to the clipboard.
-l, --list	List the matching entries on standard output.
-n, --no-clipboard	Don't copy the password of the matching entry to the clipboard, instead show the password on the terminal (by default the password is copied to the clipboard but not shown on the terminal).
-p, --password-store=DIRECTORY	Search the password store in DIRECTORY. If this option isn't given the password store is located using the $PASSWORD_STORE_DIR environment variable. If that environment variable isn't set the directory ~/.password-store is used. You can use the -p, --password-store option multiple times to search more than one password store at the same time. No distinction is made between passwords in different password stores, so the names of passwords need to be recognizable and unique.
-f, --filter=PATTERN	Don't show lines in the additional details which match the case insensitive regular expression given by PATTERN. This can be used to avoid revealing sensitive details on the terminal. You can use this option more than once.
-x, --exclude=GLOB	Ignore passwords whose name matches the given GLOB filename pattern. This argument can be repeated to add multiple exclude patterns.
-v, --verbose	Increase logging verbosity (can be repeated).
-q, --quiet	Decrease logging verbosity (can be repeated).
-h, --help	Show this message and exit.

qpass.cli.**main**()
Command line interface for the qpass program.

qpass.cli.**edit_matching_entry**(*program*, *arguments*)
Edit the matching entry.

qpass.cli.**list_matching_entries**(*program*, *arguments*)
List the entries matching the given keywords/patterns.

qpass.cli.**show_matching_entry**(*program*, *arguments*, *use_clipboard=True*, *quiet=False*, *filters=()*)
Show the matching entry on the terminal (and copy the password to the clipboard).

2.1.3 qpass.exceptions

Custom exceptions raised by *qpass*.

exception qpass.exceptions.**PasswordStoreError**
Base class for custom exceptions raised by *qpass*.

exception qpass.exceptions.**MissingPasswordStoreError**
Raised when the password store directory doesn't exist.

exception qpass.exceptions.**EmptyPasswordStoreError**
Raised when the password store is empty.

exception qpass.exceptions.**NoMatchingPasswordError**
Raised when no matching password can be selected.

CHAPTER 3

Change log

The change log lists notable changes to the project:

3.1 Changelog

The purpose of this document is to list all of the notable changes to this project. The format was inspired by Keep a Changelog. This project adheres to semantic versioning.

3.1.1 Release 2.3 (2018-12-03)

Add support for exclude lists (`qpass -x` or `qpass --exclude=GLOB`).

Explaining how I got here requires a bit of context:

- For several years now I've been using Google Authenticator for two-factor authentication (2FA) to online services like GitHub and Trello. Unfortunately Google Authenticator is quite bare bones in that it doesn't allow to export the configured 2FA accounts, which implies that switching phones requires resetting the 2FA configuration of a dozen online services...
- As a workaround you can store the "account configuration token" (the text behind the QR code that you scan) that's available when an account is configured in a secure location (explanation available here). This explains why I recently decided to reinitialize the 2FA configuration of all my online accounts (one last time) so that I can store the tokens in my password store.
- My 2FA tokens are encrypted with a separate, dedicated GPG key pair (with a stronger password) to ensure that the password to each online service is unlocked with a different secret than the 2FA token (so as not to completely undermine the second factor).
- So now whenever I run something like `qpass github` I get offered two matches and I need to make a choice, even though that choice will always be the same (the 2FA tokens are stored only as backups).
- Thanks to this qpass release I'm now able to configure the alias `qpass --exclude='*2fa*'` in my `~/.zshrc` so that I never have to be bothered by the entries containing the 2FA tokens again .

3.1.2 Release 2.2.1 (2018-06-21)

Bumped `proc` requirement to version 0.15 to pull in an upstream bug fix for hanging Travis CI builds caused by `gpg-agent` not detaching to the background properly because the standard error stream was redirected.

Lots of improvements were made to the `proc.gpg` module in proc release 0.15 and I consider the GPG agent functionality to be *quite* relevant for `qpass`, so this warrants a bug fix release.

3.1.3 Release 2.2 (2018-04-26)

- Added this changelog.
- Added `license` key to `setup.py` script.

3.1.4 Release 2.1 (2018-01-20)

The focus of this release was on hiding of sensitive details (fixes #1):

- Made `qpass --quiet` hide password entry details (related to #1).
- Made `qpass -f ignore ...` hide specific details (related to #1).
- Shuffled text processing order in format_entry()
- Included documentation in source distributions.

3.1.5 Release 2.0.2 (2017-11-20)

Bug fix for default password store discovery in CLI.

3.1.6 Release 2.0.1 (2017-07-27)

Minor bug fixes (update `__all__`, fix heading in `README.rst`).

3.1.7 Release 2.0 (2017-07-27)

Added support for multiple password stores.

3.1.8 Release 1.0.3 (2017-07-18)

Bug fix for previous commit :-).

3.1.9 Release 1.0.2 (2017-07-18)

Bug fix: Don't print superfluous whitespace for 'empty' entries.

3.1.10 Release 1.0.1 (2017-07-16)

Bug fix: Ignore failing `tty` commands.

3.1.11 Release 1.0 (2017-07-16)

Initial commit and release.

Python Module Index

q

Index