papis Documentation

Release 0.12

Alejandro Gallo

May 23, 2022

CONTENTS

Papis is a command-line based document and bibliography manager. Its command-line interface (*CLI*) is heavily tailored after Git.

CHAPTER

ONE

QUICK START

This is a tutorial that should be enough to get you started using papis. Papis tries to be as simple and lightweight as possible, therefore its document model should be too as simple as possible.

But before taking a look at its database structure let us show the daily usage of papis for a regular user. This tutorial is command-line based, so you should be familiar with opening a terminal window on your system and do some general operations with it, like creating folders and files.

1.1 Creating a new library

We will illustrate the process by creating a first library with a couple of `pdf` documents in it. Papis can be highly configured using configuration files. Many programs use configuration files maybe without you being aware of it. Papis' configuration files are stored together inside the folder

```
~/.config/papis
```

Bear in mind that ~ means "Home Directory". Inside this directory a configuration file is found,

```
~/.config/papis/config
```

Right now we will open this file for editing and we will create a library. In papis everything should be human-readable and human-editable. So adding a library is as easy as adding two lines to this configuration file.

Say that you want to create a "papers" library, where you can finally order all those pdf's hanging around on your computer. We create this library by putting these two lines inside the config file:

```
[papers]
dir = ~/Documents/mypapers
```

In the above lines we have created a library with the name `papers` which is located in the directory `~/Documents/mypapers`. So all the documents that we will be adding to the library will be located inside `~/Documents/mypapers`, and nowhere else. Everything that papis needs to take care of your `papers` library is inside the `~/Documents/mypapers` directory, self-contained.

If you have not already, add the two lines to the `~/.config/papis/config` file and save it, and we will proceed to add some documents. Of course, you have to make sure that the folder `~/Documents/mypapers` exists, so go ahead and create it

```
mkdir -p ~/Documents/mypapers
```

1.2 Adding the first document

If you don't have any special pdf lying around let me choose one for you: link. You can download this document and we are going to add it into the `papers` library.

Assuming that you have the document in the current directory and you have renamed the document to `document.pdf`, do the following to add the pdf into your library:

```
papis add document.pdf --set author "Newton" --set title "Principia Mathematica"
```

And it's done! We have added our first book to the library.

Let us see how this works exactly. Papis consists of many commands, and one of these commands is `add`. `add` itself has many flags, which are options for the given command. In the example above we have used the flags `author` and `title` to tell papis to use `Newton` as the author's name and `Principia Mathematica` as the document's title. You can see all the possible flags for the command `add` if you use the `help` flag, i.e., if you issue the following command

```
papis add --help
```

Now you are asking yourself, what happened to the pdf-file? Where is it stored? Is it stored in an obscure database somewhere in my computer? No, papis just copied the `document.pdf` file into a folder inside the library folder `~/Documents/papers/`. If you now go there, you will see that a folder with a weird name has been created. Inside of the folder there is the `document.pdf` file and another file, `info.yaml`.

If you open the `info.yaml` file you will see the following contents:

```
author: Newton
title: Principia Mathematica
files:
- document.pdf
```

This file is all that papis uses to store the information of your newly added document. It is stored in a nicely readable yaml format.

Now you already have your first document, and.. you can open it! Just do

```
papis open
```

and the document should open in your default pdf-viewer. You can change the default pdf-viewer in your configuration file (see section on *Configuration file*).

Now you can try to repeat the same process with another pdf-file lying around. When you hit `papis open` again, it will ask you which one you want. If you input parts of the title or the author's name it will try to match what you typed with the paper you are looking for, so that you can get the desired paper very easily.

Of course papis shines really in other areas, for instance imagine you are browsing this paper prl paper and you want to add it to your library, as of version `v0.9` you can issue one of these commands

```
papis add https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.124.171801/
papis add --from doi 10.1103/PhysRevLett.124.171801/
```

Here you can see it in action using the smart matching first alternative

CHAPTER

TWO

INSTALLATION

2.1 Using pip

The easiest way of installing papis is using the `PyPi` repositories and the package manager `pip3`, just open a terminal and type in

```
pip3 install papis
```

If you are on GNU/Linux like systems you might need to type `sudo`

```
sudo pip3 install papis
```

or if you prefer installing it locally then simply type

```
pip3 install --user papis
```

You can also **update** papis with `pip`

```
pip3 install --upgrade papis
```

2.2 Archlinux

- The package *papis* is also found in the archlinux repositories here.
- If you want to use the git version of `papis` instead refer to papis-git package. Thanks Julian!.

2.3 NixOS

If you are running NixOS or you have the nix package manager installed, you can install papis by running:

```
nix-env -i papis
```

If you like papis, just clone the repository

```
git clone git@github.com:papis/papis.git
cd papis
```

and start hacking it with:

```
nix-shell --expr 'with import <nixpkgs> {}; papis.overrideDerivation (drv: { src = ./.
↪; })'
```

This command will provide you a shell with all the dependencies required.

2.4 From source

First of all you have to get the code, open a terminal and hit

```
git clone https://github.com/papis/papis.git
```

or download the zip file.

Go inside of the `papis` source folder and you can either use the `Makefile` or install it with `python3`.

The general command that you have to hit is by using the `setup.py` script:

```
python3 setup.py install
```

Again, if you want to install it locally because you don't have administrative rights on your computer you simply type

```
python3 setup.py install --user
```

If you want to work on the code, you can alternatively hit

```
python3 setup.py develop --user
```

Warning: If you install the package locally, the program `papis` will be installed by default into your `~/.local/bin` directory, so that you will have to set your `PATH` accordingly.

One way of doing this in `bash` shells (`Linux` and the like, also `Ubuntu` on Windows or `cygwin`) is by adding the following line to your `~/.bashrc` file

```
export PATH=$PATH:$HOME/.local/bin
```

2.5 Requirements

Papis needs the following packages that are sometimes not installed with the system `python3` distribution

```
python3-setuptools
```

However if you have a general enough python distribution they should be installed.

2.6 Running tests

In order to run the necessary tests to submit a pull request, make sure that the following commands pass

```
python -m pytest papis/ tests/ --cov=papis
python -m mypy papis
python -m flake8 papis
```

for it, make sure that you have `pytest`, `flake8` and `mypy` installed.

You can make sure that you have everything you need to run the tests by doing in the root directory

```
pip install .[develop]
```

this command installs the necessary dependencies for developing and running the tests. Look inside of the file `setup.py` for further information.

You can also look at the folder `tools` for scripts used in the CI testing phase for further context.

CHAPTER

THREE

CONFIGURATION FILE

Papis uses a configuration file in *INI* format.

The basic configuration unit is a library. Imagine you want to have a library called `papers` and another called `books`. You can have these libraries work independently from each other.

You would declare these libraries telling papis where the folders are in your system, like so:

```
# my library for papers and stuff
[papers]
dir = ~/Documents/papers

# my library for books and related documents
[books]
dir = ~/Documents/books
```

One important aspect of the configuration system is that you can override settings on a per library basis, this means that you can set settings that should have a value for the library `papers` and another value if you're currently using the library `books`. The settings have to be set in the section under the library definition. For example, let's suppose you want to open your documents in `papers` using the pdf reader `okular` however in `books` you want to open the documents in `firefox`, for some reason, the you would write

```
# my library for papers and stuff
[papers]
dir = ~/Documents/papers
opentool = okular

# my library for books and related documents
[books]
dir = ~/Documents/books
opentool = firefox

[settings]
opentool = evince
default-library = papers
```

Here we wrote also the special section `[settings]`, which sets global settings that are valid in all libraries. Of course, every setting set within `[settings]` can be overridden by any library through the mechanism previously discussed.

A more complete example of a configuration file is the following

```
#
# This is a general section, the settings set here will be set for
# all libraries
```

(continues on next page)

(continued from previous page)

```
#
[settings]
#
# General file opener program, rifle is a nice python program
# If you're on macOS, you can write "open", if you're on linux
# you can also write "xdg-open", on windows-cygwin, you can set it to
# "cygstart"
#
opentool = rifle
# Use ranger as a file browser, a nice python program
file-browser = ranger
# Ask for confirmation when doing papis add
add-confirm = True
# Edit the info.yaml file before adding a doc into the library
# papis add --edit
add-edit = True
# Open the files before adding a document into the library
# papis add --open
add-open = True
#
# Define custom default match and header formats
#
match-format = {doc[tags]}{doc.subfolder}{doc[title]}{doc[author]}{doc[year]}
#
# Define header format with colors and multiline support
#
header-format = <red>{doc.html_escape[title]}</red>
  <span color='#ff00ff'>  {doc.html_escape[author]}</span>
  <yellow>   ({doc.html_escape[year]})</yellow>

[tui]
editmode = vi
options_list.selected_margin_style = bg:ansigreen fg:ansired
options_list.unselected_margin_style =

# Define a lib
[papers]
dir = ~/Documents/papers

# override settings from the section tui only for the papers library
# you have to prepend "tui-" to the settings
tui-editmode = emacs
tui-options_list.unselected_margin_style = bg:blue
# use whoosh as a database for papers
database-backend = whoosh
# rename files added by author and title
add-file-name = {doc[author]}{doc[title]}

# Define a lib for books
[books]
dir = ~/Documents/books
database-backend = whoosh

# Define a lib for Videos
[videos]
dir = ~/Videos/courses
```

(continues on next page)

(continued from previous page)

```
# Define a lib for contacts, why not?
# To make it work you just have to define some default settings
[contacts]
dir = ~/contacts/general
database-backend = papis
mode = contact
header-format = {doc[first_name]} {doc[last_name]}
match-format = {doc[org]} {doc[first_name]} {doc[last_name]}
browse-query-format = {doc[first_name]} {doc[last_name]}
add-open = False
```

3.1 Local configuration files

Papis also offers the possibility of creating local configuration files. The name of the local configuration file can be configured with the `local-config-file` setting.

The local configuration file can be found in the current directory of where you are issuing the papis command or in the directory of the library that you are considering in the papis command.

For instance let us suppose that you are in some project folder `~/Documents/myproject` and you have a local config file there with a definition of a new library. Then whenever you are in the `~/Documents/myproject` directory papis will also read the local configuration file found there.

On the other hand, also if you have a configuration file in the library folder for your papers, for instance in

```
~/Documents/papers/.papis.config
```

then every time that you use this library papis will also source this configuration file.

An example of a project using a local configuration file can be seen here , where the repository includes documents for component datasheets and every time `papis` is using that library the `.papis.config` file is also read and some settings will be getting overridden.

3.2 Python configuration file

For some users it would be useful to have a python file that gets loaded together with the usual configuration file, this file lives in your papis configuration directory with the name `config.py`, for instance for most users it will be in

```
~/.config/papis/config.py
```

3.3 General settings

local-config-file (*config-settings-local-config-file*)

- **Default**: `.papis.config`

Name AND relative path of the local configuration file that papis will additionally read if the file is present in the current directory or in the base directory of a given library.

This is useful, for instance, if you have a library somewhere for which you want special configuration settings but do not want these settings to cluster in your configuration file. It is also useful if you're sharing a library with someone

else and you want them to have the same settings in that library as you. Imagine you're sharing a library of datasheets with your friend Fulano. You have your library at

```
~/Documents/lib-with-fulano
```

and you've set a local configuration file there

```
~/Documents/lib-with-fulano/.papis.config
```

then whenever Fulano uses that library and the file is also present, his papis program will also read the configuration settings at the path above.

dir-umask (*config-settings-dir-umask*)

- **Default**: `493`

This is the default `umask` that will be used to create the new documents' directories.

use-git (*config-settings-use-git*)

- **Default**: `False`

Some commands will issue git commands if this option is set to `True`. For example in `mv` or `rename`.

browse-query-format (*config-settings-browse-query-format*)

- **Default**: `{doc[title]} {doc[author]}`

The query string that is to be searched for in the `browse` command whenever a search engine is used.

search-engine (*config-settings-search-engine*)

- **Default**: `https://duckduckgo.com`

Search engine to be used by some commands like `browse`.

user-agent (*config-settings-user-agent*)

- **Default**: `Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3)`

User agent used by papis whenever it obtains information from external servers.

scripts-short-help-regex (*config-settings-scripts-short-help-regex*)

- **Default**: `.*papis-short-help:  *(.*)`

This is the format of the short help indicator in external papis commands.

info-name (*config-settings-info-name*)

- **Default**: `info.yaml`

The default name of the information files.

doc-url-key-name (*config-settings-doc-url-key-name*)

- **Default**: `doc_url`

Some documents might have, apart from an url, also a file url associated with them. The key name appearing in the information file is defined by this setting.

default-library (*config-settings-default-library*)

- **Default**: `papers`

The name of the library that is to be searched when `papis` is run without library arguments.

format-doc-name (*config-settings-format-doc-name*)

- **Default**: `doc`

This setting controls the name of the document in the papis format strings like in format strings such as `match-format` or `header-format`. For instance, if you are managing videos, you might want to set this option to `vid` in order to set the `header-format` to `{doc[title]} - {doc[director]} - {doc[duration]}`.

match-format (*config-settings-match-format*)

- **Default**: `{doc[tags]}{doc.subfolder}{doc[title]}{doc[author]}{doc[year]}`

Default format that is used to match a document against in order to select it. For example if the `match-format` is equal to `{doc[year]} {doc[author]}` then the title of a document will not work to match a document, only the year and author.

header-format (*config-settings-header-format*)

- **Default:**

```
<ansired>{doc.html_escape[title]}</ansired>
 <ansigreen>{doc.html_escape[author]}</ansigreen>
  <ansiblue>({doc.html_escape[year]})</ansiblue> [<ansiyellow>{doc.html_
↪escape[tags]}</ansiyellow>]
```

Default format that is used to show a document in order to select it.

header-format-file (*config-settings-header-format-file*)

- **Default**: `None`

This option should have the path of a file with the `header-format` template. Sometimes templates can get big so this is a way of not cluttering the config file with text.

As an example you would set

```
[papers]

header-format-file = ~/.papis/config/styles/header.txt
```

info-allow-unicode (*config-settings-info-allow-unicode*)

- **Default**: `True`

This flag is to be set if you want to allow unicode characters in your info file or not. If it is set to false then a representation for the unicode characters will be written in its place. Since we should be living in an unicode world, it is set to `True` by default.

3.4 Tools options

opentool (*config-settings-opentool*)

- **Default**: `xdg-open`

This is the general program that will be used to open documents. As for now papis is not intended to detect the type of document to be opened and decide upon how to open the document. You should set this to the right program for the tool. If you are on linux you might want to take a look at ranger or let the default handle it in your system. For mac users you might set this to `open`.

browser (*config-settings-browser*)

- **Default**: `$BROWSER`

Program to be used for opening websites, the default is the environment variable `$BROWSER`.

picktool (*config-settings-picktool*)

- **Default**: `papis`

This is the program used whenever papis asks you to pick a document or options in general.

Only option is:

- papis

editor (*config-settings-editor*)

- **Default**: `$EDITOR`

Editor used to edit files in papis, e.g., for the `papis edit` command. It defaults to the `$EDITOR` environment variable, if this is not set then it will default to the `$VISUAL` environment variable. Otherwise the default editor in your system will be used.

file-browser (*config-settings-file-browser*)

- **Default**: `xdg-open`

File browser to be used when opening a directory. It defaults to the default file browser in your system, however, you can set it to different file browsers such as `dolphin`, `thunar` or `ranger` just to name a few.

3.5 Bibtex options

bibtex-journal-key (*config-settings-bibtex-journal-key*)

- **Default**: `journal`

Journal publishers may request abbreviated journal titles. This option allows the user to set the key for the journal entry when using `papis export --bibtex`.

Set as `full_journal_title` or `abbrev_journal_title` for whichever style required. Default is `journal`.

extra-bibtex-keys (*config-settings-extra-bibtex-keys*)

- **Default**: `[]`

When exporting documents in bibtex format, you might want to add non-standard bibtex keys such as `doc_url` or `tags`. You can add these as a valid python list of strings, for instance:

```
[mylib]
extra-bibtex-keys = ["tags", "doc_url"]
```

extra-bibtex-types (*config-settings-extra-bibtex-types*)

- **Default**: `[]`

Allow non-standard bibtex types to be recognized, e.g,

```
[mylib]
extra-bibtex-types = ["wikipedia", "video", "song"]
```

See bibtex reference.

multiple-authors-format (*config-settings-multiple-authors-format*)

- **Default**: `{au[family]}, {au[given]}`

When retrieving automatic author information from services like crossref.org, papis usually builds the `author` field for the given document. The format how every single author name is built is given by this setting, for instance you could customize it by the following:

```
multiple-authors-format = {au[surname]} -- {au[given_name]}
```

which would given in the case of Albert Einstein the string `Einstein -- Albert`.

multiple-authors-separator (*config-settings-multiple-authors-separator*)

- **Default**: `` and ``

Similarly to `multiple-authors-format`, this is the string that separates single authors in the `author` field. If it is set to `` and `` then you would have `<author 1> and <author 2> and ....` in the `author` field.

bibtex-unicode (*config-settings-bibtex-unicode*)

- **Default**: `False`

Whether or not to allow direct unicode characters in the document fields to be exported into the bibtex text.

3.6 `papis add` options

ref-format (*config-settings-ref-format*)

- **Default**: `{doc[title]:.15} {doc[author]:.6} {doc[year]}`

This flag is set to change the `ref` flag in the info.yaml file when a document is imported. For example: I prefer the format FirstAuthorYear e.g. Plews2019. This would be achieved by the following:

```
ref-format = {doc[author_list][0][surname]}{doc[year]}
```

In general however I recomment the default behaviour of just using the `author` key of the document, i.e.,

```
ref-format = {doc[title]:.15} {doc[author]:.6} {doc[year]}
```

The spaces in the value of the format will be important in order to capitalize the string, i.e., if you have a title like `STUDIES ABOUT EARTH AND HIMMEL` and and an author list like `mesh-ki-ang-nuna` then the built reference will be `StudiesAboutEMeshKi`.

Note: Special characters will be replaced when generating the `ref` entry (e.g. `Ö` → `O`, `.` and other symbols will be striped from the string).

If you want to add some punctuation, dots (`.`) and underscores (_) can be escaped by a backslash. For example,

```
ref-format = {doc[author_list][0][surname]}\.{doc[year]}
```

would result in 'Plews.2019'. To ensure correct capitalization you might consider inserting whitespaces after an escaped character.

add-confirm (*config-settings-add-confirm*)

- **Default**: `False`

If set to `True`, every time you run `papis add` the flag `--confirm` will be added automatically. If is set to `True` and you add it, i.e., you run `papis add --confirm`, then it will have the contrary effect, i.e., it will not ask for confirmation.

add-folder-name (*config-settings-add-folder-name*)

- **Default**: `empty string`

Default name for the folder of newly added documents. For example, if you want the folder of your documents to be named after the format `author-title` then you should set it to the papis format: `{doc[author]}-{doc[title]}`. Per default a hash followed by the author name is created.

add-file-name (*config-settings-add-file-name*)

- **Default**: `None`

Same as `add-folder-name`, but for files, not folders. If it is not set, the names of the files will be cleaned and taken *as-is*.

add-interactive (*config-settings-add-interactive*)

- **Default**: `False`

If set to `True`, every time you run `papis add` the flag `--interactive` will be added automatically. If is set to `True` and you add it, i.e., you run `papis add --interactive`, then it will have the contrary effect, i.e., it will not run in interactive mode.

add-edit (*config-settings-add-edit*)

- **Default**: `False`

If set to `True`, every time you run `papis add` the flag `--edit` will be added automatically. If it is set to `True` and you add something, i.e., you run `papis add --edit`, then it will have the contrary effect, i.e., it will not prompt to edit the info file.

add-open (*config-settings-add-open*)

- **Default**: `False`

If set to `True`, every time you run `papis add` the flag `--open` will be added automatically. If it is set to `True` and you add something, i.e., you run `papis add --open`, then it will have the contrary effect, i.e., it will not open the attached files before adding the document to the library.

3.7 `papis browse` options

browse-key (*config-settings-browse-key*)

- **Default**: `url`

This command provides the key that is used to generate the url. For users that run `papis add --from-doi`, setting browse-key to `doi` constructs the url from dx.doi.org/DOI, providing a much more accurate url.

Default value is set to `url`. If you need functionality with the `search-engine` option, set the option to an empty string e.g.

```
browse-key = ''
```

3.8 `papis edit` options

notes-name (*config-settings-notes-name*)

- **Default**: `notes.tex`

In `papis edit` you can edit notes about the document. `notes-name` is the default name of the notes file, which by default is supposed to be a TeX file. The `notes-name` is formated by the `formater`, so that the filename of notes can be dynamically defined, e.g.:

```
notes-name = notes_{doc[title]:.15}.tex
```

notes-template (*config-settings-notes-template*)

- **Default**: ````

When editing notes for the first time, a preliminary note will be generated based on a template. The path to this template is specified by `notes-template`. The template will then be formated by `formater`. This can be useful to enforce the same style in the notes for all documents.

Default value is set to `""`, which will return an empty notes file. If no file is found at the path to the template, then also an empty notes file will be generated.

3.9 Marks

open-mark (*config-settings-open-mark*)

- **Default**: `False`

If this option is set to `True`, every time papis opens a document it will ask to open a mark first. If it is set to `False`, then doing

```
papis open --mark
```

will avoid opening a mark.

mark-key-name (*config-settings-mark-key-name*)

- **Default**: `marks`

This is the default key name for the marks in the info file. For example, if you set `mark-key-name = bookmarks` then you would have in your `info.yaml` file

```
author: J. Krishnamurti
bookmarks:
- name: Chapter 1
  value: 120
```

mark-format-name (*config-settings-mark-format-name*)

- **Default**: `mark`

This is the name of the mark to be passed to the options `mark-header-format` etc... E.g. if you set `mark-format-name = m` then you could set `mark-header-format = {m[value]} - {m[name]}`.

mark-header-format (*config-settings-mark-header-format*)

- **Default**: `{mark[name]} - {mark[value]}`

This is the format in which the mark will appear whenever the user has to pick one. You can change this in order to make `marks` work in the way you like. Per default it is assumed that every mark has a `name` and a `value` key.

mark-match-format (*config-settings-mark-match-format*)

- **Default**: `{mark[name]} - {mark[value]}`

Format in which the mark name has to match the user input.

mark-opener-format (*config-settings-mark-opener-format*)

- **Default**: `xdg-open`

Due to the difficulty to generalize opening a general document at a given bookmark, the user should set this in whichever way it suits their needs. For example

- If you are using the pdf viewer `evince` and you want to open a mark, you would use

```
mark-opener-format = evince -p {mark[value]}
```

- If you are using `okular` you would use

```
mark-opener-format = okular -p {mark[value]}
```

- If you are using `zathura`, do

```
mark-opener-format = zathura -P {mark[value]}
```

3.10 Downloaders

downloader-proxy (*config-settings-downloader-proxy*)

- **Default**: `None`

There is the possibility of download papers using a proxy. To know more you can checkout this link.

3.11 Databases

default-query-string (*config-settings-default-query-string*)

- **Default**: `.`

This is the default query that a command will take if no query string is typed in the command line. For example this is the query that is passed to the command `open` whenever no search string is typed:

```
papis open
```

Imagine you want to open all papers authored by `John Smith` whenever you do not specify an input query string, i.e., `papis open`. Then setting

```
default-query-string = author:"John Smith"
```

would do the trick. Notice that the current example has been done assuming the `database-backend = papis`.

database-backend (*config-settings-database-backend*)

- **Default**: `papis`

The backend to use in the database. As for now papis supports the own database system `papis` and whoosh.

use-cache (*config-settings-use-cache*)

- **Default**: `True`

Set to `False` if you do not want to use the `cache` for the given library. This is only effective if you're using the `papis` database-backend.

cache-dir (*config-settings-cache-dir*)

- **Default**: `$XDG_CACHE_HOME`

whoosh-schema-fields (*config-settings-whoosh-schema-fields*)

- **Default**: `['doi']`

Python list with the `TEXT` fields that should be included in the whoosh database schema. For instance, say that you want to be able to search for the `doi` and `ref` of the documents, then you could include

```
whoosh-schema-fields = ['doi', 'ref']
```

whoosh-schema-prototype (*config-settings-whoosh-schema-prototype*)

- **Default:**

```
{
"author": TEXT(stored=True),
"title": TEXT(stored=True),
"year": TEXT(stored=True),
"tags": TEXT(stored=True),
}
```

This is the model for the whoosh schema, check the documentation for more information.

3.12 Terminal user interface (picker)

These options are for the terminal user interface (tui). They are defined in the section `tui` which means that you can set them in your configuration file globally like

```
[tui]
status_line_format = "F1: Help"
...
```

or inside the library sections prepending a `tui-`,

```
[papers]
tui-status_line_format = "Library papers**
...
```

status_line_format (*config-tui-status_line_format*)

- **Default**: ``{selected_index}/{number_of_documents} F1:help c-l:redraw ``

This is the format of the string that appears at the bottom in the status line. Right now there are only two variables defined:

- `selected_index`
- `number_of_documents`

Which are self-explanatory.

status_line_style (*config-tui-status_line_style*)

- **Default**: `bg:ansiwhite fg:ansiblack`

The style the status line should have. Examples are `fg:#ff00aa bg:black` etc... More information can be found here .

message_toolbar_style (*config-tui-message_toolbar_style*)

- **Default**: `bg:ansiyellow fg:ansiblack`

The style of the message toolbar, this toolbar is the one where messages of the `echo` command are rendered for instance.

options_list.selected_margin_style (*config-tui-options_list.selected_margin_style*)

- **Default**: `bg:ansiblack fg:ansigreen`

Style of the margin of the selected document in the picker.

options_list.unselected_margin_style (*config-tui-options_list.unselected_margin_style*)

- **Default**: `bg:ansiwhite`

Style of the margin of the unselected documents in the picker. If you don't want any coloring for them you can just set this setting to the empty string as such

```
tui-options_list.unselected_margin_style =
```

error_toolbar_style (*config-tui-error_toolbar_style*)

- **Default**: `bg:ansired fg:ansiblack`

The style for the error messages.

editmode (*config-tui-editmode*)

- **Default**: `emacs`

Whenever the user is typing text, one can use either `emacs` like keybindings or `vi`. If this does not tell you anything, you can just leave it as is.

move_down_key (*config-tui-move_down_key*)

- **Default**: `down`

move_up_key (*config-tui-move_up_key*)

- **Default**: `up`

move_down_while_info_window_active_key (*config-tui-move_down_while_info_window_active_key*)

- **Default**: `c-n`

move_up_while_info_window_active_key (*config-tui-move_up_while_info_window_active_key*)

- **Default**: `c-p`

focus_command_line_key (*config-tui-focus_command_line_key*)

- **Default**: `tab`

edit_document_key (*config-tui-edit_document_key*)

- **Default**: `c-e`

open_document_key (*config-tui-open_document_key*)

- **Default**: `c-o`

show_help_key (*config-tui-show_help_key*)

- **Default**: `f1`

show_info_key (*config-tui-show_info_key*)

- **Default**: `s-tab`

go_top_key (*config-tui-go_top_key*)

- **Default**: `home`

go_bottom_key (*config-tui-go_bottom_key*)

- **Default**: `end`

mark_key (*config-tui-mark_key*)

- **Default**: `c-t`

3.13 FZF integration

From version 0.12 papis ships with an *out-of-the-box* fzf integration for the picker. A minimal terminal user interface is provided and together with options for its customization. You can set the picktool to `fzf` by setting

```
picktool = fzf
```

in the configuration section of your library.

In comparison to the *built-in* papis tui the advantage of the fzf picker is that it is much faster, however a disadvantage is that it is restricted to one-line entries. Also it is important to notice that `fzf` will **only** match against what is shown on the terminal screen, as opposed to the papis matcher, that can match against the **whole** title and **whole** author text since this is controlled by the `match-format` setting. However, for many uses it might not bother the user to have this limitation of fzf.

fzf-binary (*config-settings-fzf-binary*)

- **Default**: `fzf`

Path to or name of the fzf binary.

fzf-extra-flags (*config-settings-fzf-extra-flags*)

- **Default**: `['--ansi', '--multi', '-i']`

Extra flags to be passed to fzf every time it gets called.

fzf-extra-bindings (*config-settings-fzf-extra-bindings*)

- **Default**: `['ctrl-s:jump']`

Extra bindings to fzf as a python list. Refer to the fzf documentation for more details.

fzf-header-format (*config-settings-fzf-header-format*)

- **Default**: `{c.Fore.MAGENTA}{doc[title]:<70.70}{c.Style.RESET_ALL} :: {c.Fore.CYAN}{doc[author]:<20.20}{c.Style.RESET_ALL}{c.Fore.YELLOW}«{doc[year]:4}»{c.Style.RESET_ALL}:{doc[tags]}`

Format for the entries for fzf. Notice that if you want colors you should have in `fzf-extra-flags` the `--ansi` flag and include the colors in the header-format as `ansi` escape sequences.

The papis format string is given the additional variable `c` which contains the package `colorama` in it. Refer to the `colorama` documentation to see which colors are available here. For instance, if you want the title in red you would put in your `fzf-header-format`

```
"{c.Fore.RED}{doc[title]}{c.Style.RESET_ALL}"
```

3.13.1 `fzf` with a preview window

`fzf` has the disadvantage that it does not support multiline output and it matches only against what it shows on the screen.

You can go around this issue by composing an `fzf` customization. The following configuration

```
fzf-extra-flags = ["--ansi", "--multi", "-i",
                   "--preview", "echo {} | sed -r 's/~~/\\n/g; /^ *$/d' ",
                   "--preview-window", "bottom:wrap:20%%",
                   "--color", "preview-fg:#F6E6E4,preview-bg:#5B6D5B"]

fzf-extra-bindings = ["ctrl-s:jump",
                      "ctrl-t:toggle-preview"]

fzf-header-format = {c.Fore.MAGENTA}{doc[title]}{c.Style.RESET_ALL}~~ {c.Fore.CYAN}
↪{doc[author]}{c.Style.RESET_ALL}~~ {c.Fore.YELLOW}«{doc[year]}»{c.Style.RESET_ALL}~~
↪ {c.Fore.YELLOW}{doc[journal]}{c.Style.RESET_ALL}~~ :{doc[tags]}
```

will have unrestricted titles, author, journal etc fields against which the query will match and it will show in the `fzf` preview window a tidy description of the currently selected field by replacing the token `~~` by a newline. You can try this out and play with `fzf` customizations. Please note that `bottom:wrap:20%%` has two `%` since the config file interpolator uses `%` as a reserved symbol, so it must be escaped by writing two of them.

3.14 Other

unique-document-keys (*config-settings-unique-document-keys*)

- **Default**: `['doi','ref','isbn','isbn10','url','doc_url']`

Whenever you add a new document, papis tries to figure out if you have already added this document before. This is partially done checking for some special keys, and checking if they match. Which keys are checked against is decided by this option, which should be formatted as a python list, just as in the default value.

For instance, if you add a paper with a given `doi`, and then you add another document with the same `doi`, then papis will notify you that there is already another document with this `doi` because the `doi` key is part of the `unique-document-keys` option.

document-description-format (*config-settings-document-description-format*)

- **Default**: `{doc[title]} - {doc[author]}`

`papis` sometimes will have to tell you which document it is processing through text, for instance, imagine you are updating a document

```
author: Albert Einstein
title: General Relativity
```

and papis is doing something with it. Then if your `document-description-format` is set to `{doc[title]} - {doc[author]}`, you will see that papis tells you

```
.....
Updating 'General Relativity - Albert Einstein'
...
```

so you will know exactly what is going on.

sort-field (*config-settings-sort-field*)

- **Default**: `None`

As of version `0.10`, some command line commands have the `--sort` option to sort the documents according to a given field. If you set `sort-field` in your configuration file, this will sort by default the documents according to this sort field. For instance, if you want your documents by default to be sorted by `year`, you would set `sort-field = year`.

time-stamp (*config-settings-time-stamp*)

- **Default**: `True`

Whether or not to add a timestamp to a document when is being added to papis. If documents have a timestamp, then they will be sortable using *–sort time-added* option.

formater (*config-settings-formater*)

- **Default**: `python`

The formatting language in python can be configured through plugins.

class `papis.format.`**PythonFormater**
Construct a string using a pythonic format string and a document. You can activate this formatter by setting `formater = python`.

class `papis.format.`**Jinja2Formater**
Construct a Jinja2 formated string. You can activate this formatter by setting `formater = jinja2`.

CHAPTER

FOUR

THE `INFO.YAML` FILE

At the heart of papis there is the information file. The info file contains all information about the documents.

It uses the yaml syntax to store information, which is a very human-readable language. It is quite format-free: *papis* does not assume that any special information should be there. However it will interpret the field `files` as the files linked to the document for the `papis open` command. The `files` field should be formatted as a `yaml` list.

For instance, if are storing papers with papis, then you most probably would like to store author and title in there like this:

```
author: Isaac Newton
title: Opticks, or a treatise of the reflections refractions, inflections and
  colours of light
files:
  - document.pdf
```

Here we have used the `files` field to tell papis that the paper has a pdf document attached to it. You can of course attach many other documents so that you can open them when you are opening it with the `papis open` command. For instance if you have a paper with supporting information, you could store it like such

```
author: Isaac Newton
title: Opticks, or a treatise of the reflections refractions, inflections and
  colours of light
files:
  - document.pdf
  - supporting-information.pdf
```

Therefore, in the folder where this document lives we have the following structure

```
.
└── paper-folder
    ├── info.yaml
    ├── document.pdf
    └── supporting-information.pdf
```

CHAPTER

FIVE

THE LIBRARY STRUCTURE

One of the things that makes papis interesting is the fact that its library structure is nearly nonexistent.

A papis library is linked to a directory, where all the documents are (and possibly sublibraries). What papis does is simply to go to the library folder and look for all subfolders that contain a information file, which by default is a `info.yaml` file.

Every subfolder that has an `info.yaml` file in it is a valid papis document. As an example let us consider the following library

```
/home/fulano/Documents/papers/
├── folder1
│   └── paper.pdf
├── folder2
│   ├── folder3
│   │   ├── info.yaml
│   │   └── blahblahblah.pdf
│   └── folder4
│       ├── info.yaml
│       └── output.pdf
├── classics
│   └── folder5
│       ├── info.yaml
│       └── output.pdf
├── physics
│   └── newton
│       └── principia
│           ├── document.pdf
│           ├── supplements.pdf
│           └── info.yaml
└── rpa
    └── bohm
        ├── info.yaml
        ├── notes.tex
        └── output.pdf
```

The first thing that you might notice is that there are many folders. Just to check that you understand exactly what is a document, please think about which of these pdfs is not a valid papis document... That's right!, `folder1/paper.pdf` is not a valid document since the folder1 does not contain any `info.yaml` file. You see also that it does not matter how deep the folder structure is in your library: you can have a `physics` folder in which you have a `newton` folder in which you have a folder containing the actual book `document.pdf` plus some supplementary information `supplements.pdf`. In this case, inside the `info.yaml` you would have the following `file` section

```
files:
- document.pdf
- supplements.pdf
```

which tells papis that this folder contains two relevant files.

CHAPTER

SIX

THE DATABASE

One of the things that makes papis interesting is the fact that there can be many backends for the database system, including no database.

Right now there are three types of databases that the user can use:

- **No database**

```
database-backend = papis
use-cache = False
```

- Simple cache based database - Configuration option

```
database-backend = papis
```

- **Whoosh based database.**

```
database-backend = whoosh
```

If you just plan to have up to 3000 documents in your library, you will have ample performance with the two first options. However if you're reaching higher numbers, you'll probably want to use the `Whoosh` backend for very good performance.

You can select a database by using the flag database-backend.

6.1 Papis database

The fact that there is no database means that papis should crawl through the library folder and see which folders have an `info.yaml` file, which is for slow computers (and harddrives) quite bad.

Papis implements a very rudimentary caching system. A cache is created for every library. Inside the cache the whole information already converted into python is stored.

These cache files are stored per default in

```
~/.cache/papis/
```

Notice that most papis commands will update the cache if it has to be the case. For instance the `edit` command will let you edit your document's information and after you are done editing it will update the information for the given document in the cache. If you go directly to the document and edit the info file without passing through the papis edit command, the cache will not be updated and therefore papis will not know of these changes, although they will be there. In such cases you will have to *clear the cache*.

6.1.1 Clearing the cache

To clear the cache for a given library you can use the flag `--clear-cache`, e.g.

```
papis --clear-cache
```

6.1.2 Query language

Since version *0.3* there is a query language in place for the searching of documents. The queries can contain any field of the info file, e.g., `author:einstein publisher : review` will match documents that have a matching `author` with `einstein` AND have a `publisher` matching `review`. The AND part here is important, since only the `AND` filter is implemented in this simple query language. At the moment it is not possible to do an `OR`. If you need this, you should consider using the *Whoosh database*.

For illustration, here are some examples:

- Open documents where the author key matches 'albert' (ignoring case) and year matches '19' (i.e., 1990, 2019, 1920):

  ```
  papis open 'author : albert year : 05'
  ```

- Add the restriction to the previous search that the usual matching matches the substring 'licht' in addition to the previously selected

  ```
  papis open 'author : albert year : 05 licht'
  ```

  This is not to be mixed with the restriction that the key *year* matches *'05 licht'*, which will not match any year, i.e.

  ```
  papis open 'author : albert year : "05 licht"'
  ```

6.1.3 Disabling the cache

You can disable the cache using the configuration setting `use-cache` and set it to `False`, e.g.

```
[settings]

use-cache = False

[books]
# Use cache for books but don't use for the rest of libraries
use-cache = True
```

6.2 Whoosh database

Papis has also the possibility to use the blazing fast and pure python Whoosh library. Its performance is orders of magnitude better than the crude cache based database.

Of course, the performance comes at a cost. To achieve more performance a database backend should create an index with information about the documents. Parsing a user query means going to the index and matching the query to what is found in the index. This means that the index can not in general have all the information that the info file of the documents includes.

In other words, the whoosh index will store only certain fields from the document's info files. The good news is that we can tell papis exactly which fields we want to index. These flags are

- whoosh-schema-fields
- whoosh-schema-prototype

The prototype is for advanced users. If you just want to, say, include the publisher to the fields that you can search in, then you can put

```
whoosh-schema-fields = ['publisher']
```

and you will be able to find documents by their publisher. For example, without this line set for publisher, the query

```
papis open publisher:*
```

will not return anything, since the publisher field is not being stored.

6.2.1 Query language

The whoosh database uses the whoosh query language which is much more advanced than the query language in the *Papis database*.

The whoosh query language supports both `AND` and `OR`, for instance

```
papis open '(author:einstein AND year:1905) OR title:einstein'
```

will give papers of einstein in the year 1905 together with all papers where einstein appears in the title.

You can read more about the whoosh query language here.

CHAPTER

SEVEN

COMMANDS

7.1 Add

The add command is one of the central commands of the papis command line interface. It is a very versatile command with a fair amount of options.

There are also customization settings available for this command, check out the configuration page for add.

7.1.1 Examples

- Add a document located in `~/Documents/interesting.pdf` and name the folder where it will be stored in the database `interesting-paper-2021`

  ```
  papis add ~/Documents/interesting.pdf \
      --folder-name interesting-paper-2021
  ```

  if you want to add directly some key values, like `author`, `title` and `tags`, you can also run the following:

  ```
  papis add ~/Documents/interesting.pdf \
      --folder-name interesting-paper-2021 \
      --set author 'John Smith' \
      --set title 'The interesting life of bees' \
      --set year 1985 \
      --set tags 'biology interesting bees'
  ```

- Add a paper that you have locally in a file and get the paper information through its `doi` identifier (in this case `10.10763/1.3237134` as an example):

  ```
  papis add ~/Documents/interesting.pdf --from doi 10.10763/1.3237134
  ```

- Add paper to a library named `machine-learning` from `arxiv.org`

  ```
  papis -l machine-learning add \
      --from arxiv https://arxiv.org/abs/1712.03134
  ```

- If you do not want copy the original pdfs into the library, you can also tell papis to just create a link to them, for example

  ```
  papis add --link ~/Documents/interesting.pdf \
      --from doi 10.10763/1.3237134
  ```

  will add an entry into the papis library, but the pdf document will remain at `~/Documents/interesting.pdf`, and in the document's folder there will be a link to `~/Documents/interesting.pdf` instead of the

file itself. Of course you always have to be sure that the document at `~/Documents/interesting.pdf` does not disappear, otherwise you will end up without a document to open.

- Papis also tries to make sense of the inputs that you have passed to the command, for instance you could provide only a `doi` and papis will try to know if this is indeed a `doi`

```
papis add 10.1103/PhysRevLett.123.156401
```

or from a `url`

```
papis add journals.aps.org/prl/abstract/10.1103/PhysRevLett.123.156401
papis add https://arxiv.org/abs/1712.03134
```

7.1.2 Examples in python

There is a python function in the add module that can be used to interact in a more effective way in python scripts,

papis.commands.add.**run**(*paths: List[str]*, *data: Dict[str, Any] = {}*, *folder_name: Optional[str] = None*, *file_name: Optional[str] = None*, *subfolder: Optional[str] = None*, *confirm: bool = False*, *open_file: bool = False*, *edit: bool = False*, *git: bool = False*, *link: bool = False*) → None

Parameters

- **paths** (*[]*) – Paths to the documents to be added
- **data** (*dict*) – Data for the document to be added. If more data is to be retrieved from other sources, the data dictionary will be updated from these sources.
- **folder_name** (*str*) – Name of the folder where the document will be stored
- **file_name** (*str*) – File name of the document's files to be stored.
- **subfolder** (*str*) – Folder within the library where the document's folder should be stored.
- **confirm** (*bool*) – Whether or not to ask user for confirmation before adding.
- **open_file** (*bool*) – Whether or not to ask the user for opening the file before adding.
- **edit** (*bool*) – Whether or not to ask user for editing the info file before adding.
- **git** (*bool*) – Whether or not to ask user for committing before adding, in the case of course that the library is a git repository.

7.1.3 Cli

papis add

Add a document into a given library

```
papis add [OPTIONS] [FILES]...
```

Options

-h, --help
Show this message and exit.

-s, --set <set_list>
Set some information before

-d, --subfolder <subfolder>
Subfolder in the library

--folder-name <folder_name>
Name for the document's folder (papis format)

--file-name <file_name>
File name for the document (papis format)

--from <from_importer>
Add document from a specific importer (pmid, crossref, isbn, bibtex, doi, pdf2doi, yaml, arxiv, pdf2arxivid, lib, folder)

-b, --batch
Batch mode, do not prompt or otherwise

--confirm, --no-confirm
Ask to confirm before adding to the collection

--open, --no-open
Open file before adding document

--edit, --no-edit
Edit info file before adding document

--link, --no-link
Instead of copying the file to the library, create a link toits original location

--git, --no-git
Git add and commit the new document

--list-importers, --li
List all available papis importers

Arguments

FILES
Optional argument(s)

7.2 Addto

This command adds files to existing papis documents in some library.

For instance imagine you have two pdf files, `a.pdf` and `b.pdf` that you want to add to a document that matches with the query string `einstein photon definition`, then you would use

```
papis addto 'einstein photon definition' -f a.pdf -f b.pdf
```

notice that we repeat two times the flag `-f`, this is important.

7.2.1 Cli

papis addto

Add files to an existing document

```
papis addto [OPTIONS] [QUERY]
```

Options

-h, --help
Show this message and exit.

--git, --no-git
Add and commit files

--sort <FIELD>
Sort documents with respect to FIELD

--reverse
Reverse sort order

-f, --files <files>
File fullpaths to documents

--file-name <file_name>
File name for the document (papis format)

--doc-folder <doc_folder>
Apply action to a document path

Arguments

QUERY
Optional argument

7.3 Browse

This command will try its best to find a source in the internet for the document at hand.

Of course if the document has an url key in its info file, it will use this url to open it in a browser. Also if it has a doc_url key, or a doi, it will try to compose urls out of these to open it.

If none of the above work, then it will try to use a search engine with the document's information (using the `browse-query-format`). You can select which search engine you want to use using the `search-engine` setting.

It uses the configuration option `browse-key` to form an url according to which key is given in the document. You can bypass this option using the *-k* flag issuing the command.

```
papis browse -k doi einstein
```

This will form an url through the DOI of the document.

```
papis browse -k isbn
```

This will form an url through the ISBN of the document using isbnsearch.org.

```
papis browse -k ads
```

This will form an url using the gread ADS service and there you can check for similar papers, citations, references and much more. Please note that for this to work the document should have a DOI attached to it.

```
papis browse -k whatever
```

This will consider the key `whatever` of the document to be a valid url, I guess at this point you'll know what you're doing.

```
papis browse -k search-engine
```

This is the default, it will do a search-engine search with the data of your paper and hopefully you'll find it.

7.3.1 Cli

papis browse

Open document's url in a browser

```
papis browse [OPTIONS] [QUERY]
```

Options

-h, --help
Show this message and exit.

--sort <FIELD>
Sort documents with respect to FIELD

--reverse
Reverse sort order

-k, --key <key>
Use the value of the document's key to open in the browser, e.g. doi, url, doc_url ...

-n, --print
Just print out the url, do not open it with browser

-a, --all
Apply action to all matching documents

--doc-folder <doc_folder>
Apply action to a document path

Arguments

QUERY
Optional argument

7.4 Bibtex

This command helps to interact with *bib* files in your LaTeX projects.

7.4.1 Examples

I use it for opening some papers for instance

```
papis bibtex read new_papers.bib open
```

or to add papers to the bib

```
papis bibtex          \
  read new_papers.bib \ # Read bib file
  add einstein        \ # Pick a doc with query 'einstein' from library
  add heisenberg      \ # Pick a doc with query 'heisenberg' from library
  save new_papers.bib   # Save in new_papers.bib
```

or if I update some information in my papis `yaml` files then I can do

```
papis bibtex          \
  read new_papers.bib \ # Read bib file
  update -f           \ # Update what has been read from papis library
  save new_papers.bib   # save everything to new_papers.bib, overwriting
```

7.4.2 Local configuration file

If you are working in a local folder where you have a bib file called `main.bib`, you'll grow sick and tired of writing always `read main.bib` and `save main.bib`, so you can write a local configuration file `.papis.config` for `papis bibtex` to read and write automatically

```
[bibtex]
default-read-bibfile = main.bib
default-save-bibfile = main.bib
auto-read = True
```

with this setup, you can just do

```
papis bibtex add einstein save
```

7.4.3 Check references quality

When you're collaborating with someone, you might come across malformed or incomplete references. Most journals want to have all the ``doi``s and urls available. You can automate this diagnostic with

For this you kan use the command `doctor`

```
papis bibtex read mybib.bib doctor
```

Mostly I want to have only the references in my project's bib file that are actually cited in the latex file, you can check which references are not cited in the tex files by doing

```
papis bibtex iscited -f main.tex -f chapter-2.tex
```

and you can then filter them out using the command `filter-cited`.

To monitor the health of the bib project's file, I mostly have a target in the project's `Makefile` like

```
.PHONY: check-bib
check-bib:
    papis bibtex iscited -f main.tex doctor
```

it does not solve all problems under the sun, but it is really better than no check!

7.4.4 Vim integration

Right now, you can easily use it from vim with these simple lines

```
function! PapisBibtexRef()
  let l:temp = tempname()
  echom l:temp
  silent exec "!papis bibtex ref -o ".l:temp
  let l:olda = @a
  let @a = join(readfile(l:temp), ',')
  normal! "ap
  redraw!
  let @a = l:olda
endfunction

command! -nargs=0 BibRef call PapisBibtexRef()
command! -nargs=0 BibOpen exec "!papis bibtex open"
```

And use like such:

7.4.5 Cli

papis bibtex

A papis script to interact with bibtex files

```
papis bibtex [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...
```

Options

-h, --help
Show this message and exit.

--noar, --no-auto-read
Do not auto read even if the configuration file says it

Commands

add
Add a reference to the bibtex file

bibtex
Import documents from a bibtex file Examples...

browse
browse a document in the documents list

doctor
Check bibfile for correctness, missing keys...

edit
edit a document in the documents list

filter-cited
Filter cited documents from the read bib file...

import
Import documents to papis e.g.

iscited
Check which documents are not cited e.g.

open
Open a document in the documents list

ref
Print the reference for a document

rm
Remove a document from the documents list

save
Save the documents imported in bibtex format

sort
Sort documents

unique
Remove repetitions

update
Update documents from and to the library

7.5 Config

The command config is a useful command because it allows you to check the configuration settings' values that your current *papis* session is using.

For example let's say that you want to see which `dir` setting your current library is using (i.e., the directory or the dir that appears in the definition of the library in the configuration file), then you would simply do:

```
papis config dir
```

If you wanted to see which `dir` the library `books` has, for example then you would do

```
papis -l books config dir
```

This works as well for default settings, i.e., settings that you have not customized, for example the setting `match-format`, you would check it with

```
papis config match-format
> {doc[tags]}{doc.subfolder}{doc[title]}{doc[author]}{doc[year]}
```

You can find a list of all available settings in the configuration section.

7.5.1 Cli

papis config

Print configuration values

```
papis config [OPTIONS] OPTION
```

Options

-h, --help
Show this message and exit.

Arguments

OPTION
Required argument

7.6 Main

7.6.1 Examples

- To override some configuration options, you can use the flag `--set`, for instance, if you want to override the editor used and the opentool to open documents, you can just type

```
papis --set editor gedit --set opentool firefox edit
papis --set editor gedit --set opentool firefox open
```

- If you want to list the libraries and pick one before sending a database query to papis, use `--pick-lib` as such

```
papis --pick-lib open 'einstein relativity'
```

7.6.2 Cli

papis

```
papis [OPTIONS] COMMAND [ARGS]...
```

Options

-h, --help
Show this message and exit.

--version
Show the version and exit.

-v, --verbose
Make the output verbose (equivalent to –log DEBUG)

--profile <profile>
Print profiling information into file

-l, --lib <lib>
Choose a library name or library path (unnamed library)

-c, --config <config>
Configuration file to use

--pick-lib
Pick library to use

--cc, --clear-cache
Clear cache of the library used

-s, --set <set_list>
Set key value, e.g., –set info-name information.yaml –set opentool evince

--color <color>
Prevent the output from having color

Options always | auto | no

--log <log>
Logging level

Options INFO | DEBUG | WARNING | ERROR | CRITICAL

--logfile <logfile>
File to dump the log

--np <np>
Use number of processors for multicore functionalities in papis

7.7 Edit

This command edits the information of the documents. The editor used is defined by the `editor` configuration setting.

7.7.1 Cli

papis edit

Edit document information from a given library

```
papis edit [OPTIONS] [QUERY]
```

Options

-h, --help
Show this message and exit.

--doc-folder <doc_folder>
Apply action to a document path

--git, --no-git
Add changes made to the info file

--sort <FIELD>
Sort documents with respect to FIELD

--reverse
Reverse sort order

-n, --notes
Edit notes associated to the document

-a, --all
Apply action to all matching documents

-e, --editor <editor>
Editor to be used

Arguments

QUERY
Optional argument

7.8 Explore

This command is in an experimental stage but it might be useful for many people.

Imagine you want to search for some papers online, but you don't want to go into a browser and look for it. Explore gives you way to do this, using several services available online, more should be coming on the way.

An excellent such resource is crossref, which you can use by using the subcommand crossref:

```
papis explore crossref --author 'Freeman Dyson'
```

If you issue this command, you will see some text but basically nothing will happen. This is because `explore` is conceived in such a way as to concatenate commands, doing a simple

```
papis explore crossref -h
```

will tell you which commands are available. Let us suppose that you want to look for some documents on crossref, say some papers of Schroedinger, and you want to store them into a bibtex file called `lib.bib`, then you could concatenate the commands `crossref` and `export --format bibtex` as such

```
papis explore crossref -a 'Schrodinger' export --format bibtex lib.bib
```

This will store everything that you got from crossref in the file `lib.bib` and store in bibtex format. `explore` is much more flexible than that, you can also pick just one document to store, for instance let's assume that you don't want to store all retrieved documents but only one that you pick, the `pick` command will take care of it

```
papis explore crossref -a 'Schrodinger' pick export --format bibtex lib.bib
```

notice how the `pick` command is situated before the `export`. More generally you could write something like

```
papis explore \
    crossref -a Schroedinger \
    crossref -a Einstein \
    arxiv -a 'Felix Hummel' \
    export --format yaml docs.yaml \
    pick  \
    export --format bibtex specially-picked-document.bib
```

The upper command will look in crossref for documents authored by Schrodinger, then also by Einstein, and will look on the arxiv for papers authored by Felix Hummel. At the end, all these documents will be stored in the `docs.yaml`. After that we pick one document from them and store the information in the file `specially-picked-document.bib`, and we could go on and on.

If you want to follow-up on these documents and get them again to pick one, you could use the `yaml` command to read in document information from a yaml file, i.e., the previously created `docs.yaml`

```
papis explore \
    yaml docs.yaml \
    pick \
    cmd 'papis scihub {doc[doi]}' \
    cmd 'firefox {doc[url]}'
```

In this last example, we read the documents' information from `docs.yaml` and pick a document, which then feed into the `explore cmd` command, that accepts a papis formatting string to issue a general shell command. In this case, the picked document gets fed into the `papis scihub` command which tries to download the document using `scihub`, and also this very document is tried to be opened by firefox (in case the document does have a `url`).

7.8.1 Cli

papis explore

Explore new documents using a variety of resources

```
papis explore [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...
```

Options

-h, --help
Show this message and exit.

add

```
papis explore add [OPTIONS]
```

arxiv

Look for documents on ArXiV.org.

Examples of its usage are

papis explore arxiv -a 'Hummel' -m 100 arxiv -a 'Garnet Chan' pick

If you want to search for the exact author name 'John Smith', you should enclose it in extra quotes, as in the example below

papis explore arxiv -a '"John Smith"' pick

```
papis explore arxiv [OPTIONS]
```

Options

-h, --help
Show this message and exit.

-q, --query <query>

-a, --author <author>

-t, --title <title>

--abstract <abstract>

--comment <comment>

--journal <journal>

--report-number <report_number>

--category <category>

--id-list <id_list>

--page <page>

-m, --max <max>

base

Look for documents on the BielefeldAcademicSearchEngine

Examples of its usage are

papis explore base -q 'Albert einstein' pick cmd 'firefox {doc[url]}'

```
papis explore base [OPTIONS]
```

Options

-h, --help
Show this message and exit.

-q, --query <query>

bibtex

Import documents from a bibtex file

Examples of its usage are

papis explore bibtex lib.bib pick

```
papis explore bibtex [OPTIONS] BIBFILE
```

Options

-h, --help
Show this message and exit.

Arguments

BIBFILE
Required argument

citations

Query the citations of a paper

Example:

Go through the citations of a paper and export it in a yaml file

papis explore citations 'einstein' export –format yaml einstein.yaml

```
papis explore citations [OPTIONS] [QUERY]
```

Options

--doc-folder <doc_folder>
Apply action to a document path

-h, --help
Show this message and exit.

-s, --save
Store the citations in the document's folder for later use

--rmfile
Remove the stored citations file

-m, --max-citations <max_citations>
Number of citations to be retrieved

Arguments

QUERY
Optional argument

cmd

Run a general command on the document list

Examples of its usage are:

Look for 200 Schroedinger papers, pick one, and add it via papis-scihub

papis explore crossref -m 200 -a 'Schrodinger' pick cmd 'papis scihub {doc[doi]}'

```
papis explore cmd [OPTIONS] COMMAND
```

Options

-h, --help
Show this message and exit.

Arguments

COMMAND
Required argument

crossref

Look for documents on crossref.org.

Examples of its usage are

papis explore crossref -a 'Albert einstein' pick export –bibtex lib.bib

```
papis explore crossref [OPTIONS]
```

Options

-h, --help
Show this message and exit.

-q, --query <query>
General query

-a, --author <author>
Author of the query

-t, --title <title>
Title of the query

-m, --max <_ma>
Maximum number of results

-f, --filter <_filters>
Filters to apply

-o, --order <order>
Order of appearance according to sorting

Default desc

Options asc | desc

-s, --sort <sort>
Sorting parameter

Default score

Options relevance | score | updated | deposited | indexed | published | published-print | published-online | issued | is-referenced-by-count | references-count

dissemin

Look for documents on dissem.in

Examples of its usage are

papis explore dissemin -q 'Albert einstein' pick cmd 'firefox {doc[url]}'

```
papis explore dissemin [OPTIONS]
```

Options

-h, --help
Show this message and exit.

-q, --query <query>

export

Export retrieved documents into various formats for later use

Examples of its usage are

papis explore crossref -m 200 -a 'Schrodinger' export –yaml lib.yaml

```
papis explore export [OPTIONS]
```

Options

-h, --help
Show this message and exit.

-f, --format <fmt>
Format for the document

Options yaml | json | bibtex

-o, --out <out>
Outfile to write information to

isbn

Look for documents using isbnlib

Examples of its usage are

papis explore isbn -q 'Albert einstein' pick cmd 'firefox {doc[url]}'

```
papis explore isbn [OPTIONS]
```

Options

-h, --help
Show this message and exit.

-q, --query <query>

-s, --service <service>

Options wcat | goob | openl

isbnplus

Look for documents on isbnplus.com

Examples of its usage are

papis explore isbnplus -q 'Albert einstein' pick cmd 'firefox {doc[url]}'

```
papis explore isbnplus [OPTIONS]
```

Options

-h, --help
Show this message and exit.

-q, --query <query>

-a, --author <author>

-t, --title <title>

json

Import documents from a json file

Examples of its usage are

papis explore json lib.json pick

```
papis explore json [OPTIONS] JSONFILE
```

Options

-h, --help
Show this message and exit.

Arguments

JSONFILE
Required argument

lib

Query for documents in your library

Examples of its usage are

papis lib -l books einstein pick

```
papis explore lib [OPTIONS] [QUERY]
```

Options

-h, --help
Show this message and exit.

--doc-folder <doc_folder>
Apply action to a document path

-l, --library <library>
Papis library to look

Arguments

QUERY
Optional argument

pick

Pick a document from the retrieved documents

Examples of its usage are

papis explore bibtex lib.bib pick

```
papis explore pick [OPTIONS]
```

Options

-h, --help
Show this message and exit.

-n, --number <number>
Pick automatically the n-th document

yaml

Import documents from a yaml file

Examples of its usage are

papis explore yaml lib.yaml pick

```
papis explore yaml [OPTIONS] YAMLFILE
```

Options

-h, --help
Show this message and exit.

Arguments

YAMLFILE
Required argument

7.9 Export

The export command is useful to work with other programs such as bibtex.

Some examples of its usage are:

- Export one of the documents matching the author with einstein to bibtex:

```
papis export --format bibtex 'author : einstein'
```

or export all of them

```
papis export --format bibtex --all 'author : einstein'
```

- Export all documents to bibtex and save them into a `lib.bib` file

```
papis export --all --format bibtex --out lib.bib
```

- Export a folder of one of the documents matching the word `krebs` into a folder named, `interesting-document`

```
  papis export --folder --out interesting-document krebs

this will create the folder ``interesting-document`` containing the
``info.yaml`` file, the linked documents and a ``bibtex`` file for
sharing with other people.
```

Note: Every document exported also comes with the key *_papis_local_folder* associated that points to the full local folder path where the document is stored in the file system. This is done for the convenience of third party apps.

7.9.1 Cli

papis export

Export a document from a given library

```
papis export [OPTIONS] [QUERY]
```

Options

-h, --help
Show this message and exit.

--doc-folder <doc_folder>
Apply action to a document path

--sort <FIELD>
Sort documents with respect to FIELD

--reverse
Reverse sort order

-a, --all
Apply action to all matching documents

--folder
Export document folder to share

-o, --out <out>
Outfile or outdir

-f, --format <fmt>
Format for the document

Options yaml | json | bibtex

Arguments

QUERY
Optional argument

7.10 Git

This command is useful if your library is itself a git repository. You can use this command to issue git commands in your library repository without having to change your current directory.

7.10.1 CLI Examples

- Check the status of the library repository:

```
papis git status
```

- Commit all changes:

```
papis git commit -a
```

7.11 List

This command is to list contents of a library.

7.11.1 CLI Examples

- List all document files associated will all entries:

```
papis list --all --file
```

- List all document year and title with custom formatting:

```
papis list --all --format '{doc[year]} {doc[title]}'
```

- List all documents according to the bibitem formatting (stored in a template file `bibitem.template`):

```
papis list --all --template bibitem.template
```

7.11.2 Cli

papis list

List documents' properties

```
papis list [OPTIONS] [QUERY]
```

Options

-h, --help
Show this message and exit.

--sort <FIELD>
Sort documents with respect to FIELD

--reverse
Reverse sort order

-i, --info
Show the info file name associated with the document

-f, --file
Show the file name associated with the document

-d, --dir
Show the folder name associated with the document

-n, --notes
List notes files, if any

--format <_format>
List entries using a custom papis format, e.g. '{doc[year] {doc[title]}

--template <template>
Template file containing a papis format to list entries

--downloaders
List available downloaders

--libraries
List defined libraries

-a, --all
Apply action to all matching documents

Arguments

QUERY
Optional argument

7.12 Mv

7.12.1 Cli

papis mv

Move a document into some other path

```
papis mv [OPTIONS] [QUERY]
```

Options

-h, --help
Show this message and exit.

--git, --no-git
Add git interoperability

--sort <FIELD>
Sort documents with respect to FIELD

--reverse
Reverse sort order

--doc-folder <doc_folder>
Apply action to a document path

Arguments

QUERY
Optional argument

7.13 Open

The open command is a very important command in the papis workflow. With it you can open documents, folders or marks.

7.13.1 Marks

One of special things about this command is the possibility of creating marks for documents. As you would imagine, it is in general difficult to create marks for any kind of data. For instance, if our library consists of pdf files and epub files for instance, we would like to define bookmarks in order to go back to them at some later point.

How you define marks can be customized through the marks configuration settings here. The default way of doing it is just by defining a `marks` list in a document. Let us look at a concrete example:

```
author: Isaiah Shavitt, Rodney J. Bartlett
edition: '1'
files: [book.pdf]
isbn: 052181832X,9780521818322

marks:
- {name: Intermediates definition, value: 344}
- {name: EOM equations, value: 455}

publisher: Cambridge University Press
ref: book:293288
series: Cambridge Molecular Science
title: 'Many-Body Methods in Chemistry and Physics'
type: book
year: '2009'
```

This book has defined two marks. Each mark has a name and a value. If you tell the open command to open marks, then it will look for the marks and open the value (page number). This is the default behaviour, however if you go to the configuration you'll see that you can change the convention to what it suits you.

7.13.2 Examples

- Open a pdf file linked to a document matching the string `bohm`

  ```
  papis open bohm
  ```

- Open the folder where this last document is stored

  ```
  papis open -d bohm
  ```

  Please notice that the file browser used will be also related to the file-browser setting.

- Open a mark defined in the info file

  ```
  papis open --mark bohm
  ```

7.13.3 Cli

papis open

Open document from a given library

```
papis open [OPTIONS] [QUERY]
```

Options

-h, --help
Show this message and exit.

--sort <FIELD>
Sort documents with respect to FIELD

--reverse
Reverse sort order

--doc-folder <doc_folder>
Apply action to a document path

-a, --all
Apply action to all matching documents

--tool <tool>
Tool for opening the file (opentool)

-d, --dir
Open directory

-m, --mark, --no-mark
Open mark

Arguments

QUERY
Optional argument

7.14 Rename

7.14.1 Cli

papis rename

Rename entry

```
papis rename [OPTIONS] [QUERY]
```

Options

-h, --help
Show this message and exit.

--git, --no-git
Add git interoperability

--sort <FIELD>
Sort documents with respect to FIELD

--reverse
Reverse sort order

--doc-folder <doc_folder>
Apply action to a document path

Arguments

QUERY
Optional argument

7.15 Rm

7.15.1 Cli

papis rm

Delete a document, a file, or a notes-file

```
papis rm [OPTIONS] [QUERY]
```

Options

-h, --help
Show this message and exit.

--git, --no-git
Remove in git

--sort <FIELD>
Sort documents with respect to FIELD

--reverse
Reverse sort order

--doc-folder <doc_folder>
Apply action to a document path

--file
Remove files from a document instead of the whole folder

-n, --notes
Remove the notes file from a document instead of the whole folder

-f, --force
Do not confirm removal

-a, --all
Apply action to all matching documents

Arguments

QUERY
Optional argument

7.16 Run

This command is useful to issue commands in the directory of your library.

7.16.1 CLI Examples

- List files in your directory

```
papis run ls
```

- Find a file in your directory using the `find` command

```
papis run find -name 'document.pdf'
```

- Find all pdfs in the document folders matching einstein

```
  papis run -p einstein --all -- find . -name '*.pdf'

notice that in general, the symbol ``--`` is advisable
so that the arguments after it are considered as positional arguments
for the shell commands.

In this example you could also use pipes, for instance to print the
absolute path to the files, in linux you can use the command
``readlink -f`` and a pipe ``|`` to do this, i.e.:
```

```
papis run -p einstein \
        --all -- "find . -name '*.pdf' | xargs readlink -f"
```

- Replace some text in all info.yaml files by something. For instance imagine you want to replace all `note` field names in the `info.yaml` files by `_note` so that the `note` field does not get exported to bibtex. You can do

```
papis run -a -- sed -i "s/^note:/_note:/" info.yaml
```

7.16.2 Cli

papis run

Run an arbitrary shell command in the library or command folder

```
papis run [OPTIONS] <COMMANDS>
```

Options

-h, --help
Show this message and exit.

-p, --pick <QUERY>
Give a query to pick a document to run the command in its folder

--sort <FIELD>
Sort documents with respect to FIELD

--reverse
Reverse sort order

--doc-folder <doc_folder>
Apply action to a document path

-a, --all
Apply action to all matching documents

--prefix <PREFIX>
Prefix shell commands by a prefix command

Arguments

<COMMANDS>
Optional argument(s)

7.17 Update

This command is to update the information of the documents.

Some examples of the usage are given below

7.17.1 Examples

- Update a document automatically and interactively (searching by `doi` number in *crossref*, or in other sources. . .)

  ```
  papis update --auto -i "author : dyson"
  ```

- Update your library from a bib(la)tex file where many entries are listed. papis will try to look for documents in your library that match these entries and will ask you entry per entry to update it (of course this is done if you use the `-i` flag for interactively doing it). In the example `libraryfile.bib` is a file containing many entries.

```
papis update --from bibtex libraryfile.bib -i
```

- Tag all einstein papers with the tag classics

```
papis update --all --set tags classics einstein
```

and add the tag of `physics` to all papers tagged as `classics`

```
papis update --all --set tags '{doc[tags]} physics' einstein
```

7.17.2 Cli

papis update

Update a document from a given library.

```
papis update [OPTIONS] [QUERY]
```

Options

-h, --help
Show this message and exit.

--git, --no-git
Add git interoperability

--doc-folder <doc_folder>
Apply action to a document path

-a, --all
Apply action to all matching documents

--sort <FIELD>
Sort documents with respect to FIELD

--reverse
Reverse sort order

--auto
Try to parse information from different sources

--from <from_importer>
Add document from a specific importer (pmid, crossref, isbn, bibtex, doi, pdf2doi, yaml, arxiv, pdf2arxivid, lib, folder)

-s, --set <set_tuples>
Update document's information with key value.The value can be a papis format.

Arguments

QUERY
Optional argument

CHAPTER

EIGHT

WEB APPLICATION

Since v0.12, papis ships with an experimental, simple yet handy web application.

You can start the web application by issuing the following command

```
papis serve --port 8181 --host localhost
```

Then input the following url in your browser

```
http://localhost:8181
```

This web application can be useful if you intend to read access to your documents from a tablet or from another computer.

Further documentation will be available soon, but bear in mind that this web application is experimental, bug reports and suggestions are highly appreciated.

CHAPTER

NINE

GUI

Papis is a program mainly intended for the command line, however, some experimental frontends have been conceived for it and are already downloadable from pip.

See for instance

- papis-rofi
- papis-dmenu

And more to come!

CHAPTER

TEN

HOOKS

From version `0.12` papis has a minimal hook infrastructure. Some parts of papis define and run hooks so that users and plugin writers can also tap into this functionality.

A hook is declared in the same way as a plugin, in fact they are implemented in the same way within the stevedore plugin.

10.1 1 Writing hooks as a user

Right now the only way to add a hook as a user is using your `config.py` configuration file, which gets loaded when your papis configuration gets loaded.

As an example you can add a function to the `on_edit_done` hook like

```
import papis.hooks

papis.hooks.add("on_edit_done", lambda: print(42))
```

10.2 2 Writing hooks as a developer

To add a hook as a plugin writer or a developer you can just add the `entry_point` to the `setup.py` file, for instance for the `on_edit_done` hook you would write

```
"papis.hook.on_edit_done" : [
    "my_hook_name=path.to.my:function,
]
```

CHAPTER

ELEVEN

CUSTOM SCRIPTS

As in git, you can write custom scripts to include them in the command spectrum of papis.

11.1 Example: Mail script

Imagine you want to write a script to send papers to someone via the email client `mutt` (you can try to do it with another mail client), you could write the following script called `papis-mail`:

```
#! /usr/bin/env bash
# papis-short-help: Email a paper to my friend

folder_name=$1
zip_name="${folder_name}.zip"

papis -l ${PAPIS_LIB} export --folder --out ${folder_name}
zip -r ${zip_name} ${folder_name}

mutt -a ${zip_name}
```

Papis defines environment variables such as `PAPIS_LIB` so that external scripts can make use of the user input.

To use the script you can put it somewhere in your `PATH` or alternatively inside the `~/.papis/scripts` folder. If this is the case then you can run

```
papis -h
```

and you will see that there is another command besides the default called `mail`. In fact, you will see

```
positional arguments:
  command               For further information for every command, type in 'papis
↪<command> -h'
    add                 Add a document into a given library
    ............        ........................
    mail                Email a paper to my friend

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

where the description `Email a paper to my friend` is there because we have defined the comment `# papis-short-help:  Email a paper to my friend` in the header of the script.

Then, if you type

```
papis -l mylib mail this_paper
```

this will create a folder called `this_paper` with a selection of a document, zip it and send it to whoever you choose to.

11.2 Example: Accessing papis from within mutt

You may want to pick documents to attach to your email in `mutt` from the papis interface.

Add this code to your `muttrc`

```
# # macro to attach paper from papis
macro attach,compose \cp \
"\
<enter-command>unset wait_key<enter>\                                    # Don't require
↪'press any key'
<shell-escape>rm -rf /tmp/paper /tmp/paper.zip<enter>\                   # remove the␣
↪folder /tmp/paper if it already exists
<shell-escape>papis export --folder -o /tmp/paper<enter>\                # start papis␣
↪with the --folder flag
<shell-escape>zip -r /tmp/paper.zip /tmp/paper<enter>\                   # zip the␣
↪directory
<attach-file>/tmp/paper.zip<enter>\                                      # attach zip␣
↪file to the email
"
```

Try it out with `Ctrl-p` on your `Compose` screen. This makes use of the `papis export --folder` flag that moves the paper folder you choose to a temporary location (`/tmp/paper`). Mutt will then attach the paper to the email, which you can rename to be more descriptive with `R`.

11.3 Example: Define papis mode in i3wm

This is an example of using papis with the window manager *i3*.

```
# Enter papis mode
bindsym $mod+Ctrl+p mode "papis"

# Define papis mode
mode "papis" {

  # open documents
  bindsym $mod+o exec python3 -m papis.main \
    --pick-lib --set picktool dmenu open

  # edit documents
  bindsym $mod+e exec python3 -m papis.main \
    --pick-lib --set picktool dmenu --set editor gvim edit

  # open document's url
  bindsym $mod+b exec python3 -m papis.main \
    --pick-lib --set picktool dmenu browse

  # return to default mode
```

(continues on next page)

(continued from previous page)

```
  bindsym Ctrl+c mode "default"
  bindsym Return mode "default"
  bindsym Escape mode "default"
}
```

11.4 Useful links

- **`Get paper references with papis <https://alejandrogallo.github.io/blog/get-paper-references.html`_** .. code:: sh

  citget() { query=$1 shift papis explore \

  citations -s “$query” \ pick \ cmd “papis add –from doi {doc[doi]} $@”

  }

CHAPTER

TWELVE

API

This module describes which functions are intended to be used by users to create papis scripts.

papis.api.**clear_lib_cache**(*lib: Optional[str] = None*) → None
Clear cache associated with a library. If no library is given then the current library is used.

Parameters **lib** (*str*) – Library name.

```
>>> clear_lib_cache()
```

papis.api.**doi_to_data**(*doi: str*) → Dict[str, Any]
Try to get from a DOI expression a dictionary with the document's data using the crossref module.

Parameters **doi** (*str*) – DOI expression.

Returns Document's data

Return type dict

papis.api.**edit_file**(*file_path: str*, *wait: bool = True*) → None
Edit file using the `editor` key value as a program to handle file_path.

Parameters

- **file_path** (*str*) – File path to be handled.
- **wait** (*bool*) – Wait for the completion of the opener program to continue

papis.api.**get_all_documents_in_lib**(*library: Optional[str] = None*) → List[papis.document.Document]
Get ALL documents contained in the given library with possibly.

Parameters **library** (*str*) – Library name.

Returns List of all documents.

Return type list

```
>>> import tempfile
>>> folder = tempfile.mkdtemp()
>>> set_lib_from_name(folder)
>>> docs = get_all_documents_in_lib(folder)
>>> len(docs)
0
```

papis.api.**get_documents_in_dir**(*directory: str*, *search: str = ''*) → List[papis.document.Document]
Get documents contained in the given folder with possibly a search string.

Parameters

- **directory** (*str*) – Folder path.
- **search** (*str*) – Search string

Returns List of filtered documents.

Return type list

```
>>> import tempfile
>>> docs = get_documents_in_dir(tempfile.mkdtemp())
>>> len(docs)
0
```

papis.api.**get_documents_in_lib**(*library: Optional[str] = None*, *search: str = ''*) → List[papis.document.Document]

Get documents contained in the given library with possibly a search string.

Parameters

- **library** (*str*) – Library name.
- **search** (*str*) – Search string

Returns List of filtered documents.

Return type list

papis.api.**get_lib_name**() → str

Get current library, it either retrieves the library from the environment PAPIS_LIB variable or from the command line args passed by the user.

Returns Library name

Return type str

```
>>> get_lib_name() is not None
True
```

papis.api.**get_libraries**() → List[str]

Get all libraries declared in the configuration. A library is discovered if the `dir` or `dirs` key defined in the library section.

Returns List of library names

Return type list

```
>>> len(get_libraries()) >= 1
True
```

papis.api.**open_dir**(*dir_path: str*, *wait: bool = True*) → None

Open dir using the `file-browser` key value as a program to open dir_path.

Parameters

- **dir_path** (*str*) – Folder path to be handled.
- **wait** (*bool*) – Wait for the completion of the opener program to continue

papis.api.**open_file**(*file_path: str*, *wait: bool = True*) → None

Open file using the `opentool` key value as a program to handle file_path.

Parameters

- **file_path** (*str*) – File path to be handled.

- **wait** (*bool*) – Wait for the completion of the opener program to continue

papis.api.**set_lib_from_name**(*library: str*) → None

Set current library, it either sets the library in the environment PAPIS_LIB variable or in the command line args passed by the user.

Parameters **library** (*str*) – Name of library or path to a given library

CHAPTER

THIRTEEN

PLUGIN ARCHITECTURE

13.1 General architecture

Papis uses the package stevedore for general plugin management.

The only papis module invoking `stevedore` should be `papis/plugin.py`.

The different plugins in papis like `papis.command`, `papis.exporter` etc. define a so-called `ExtensionManager` which loads various objects that have been declared in a python package somewhere.

For example, the yaml exporter is defined as

```
def exporter(documents: List[papis.document.Document]) -> str:
    string = yaml.dump_all(
        [papis.document.to_dict(document) for document in documents],
        allow_unicode=True)
    return str(string)
```

and declared in `setup.py` as

```
setup(
    ...
    entry_points={
        'papis.exporter': [
            'bibtex=papis.bibtex:exporter',
            'json=papis.json:exporter',
            'yaml=papis.yaml:exporter',
        ],
    ...
)
```

and the exporter can be used as in the code below

```
import papis.plugin
extension_manager = papis.plugin.get_extension_manager("papis.exporter")
# yaml_exporter is the function defined above
yaml_exporter = extension_manager["yaml"].plugin

yaml_string = yaml_exporter(mydocs)
```

Now a developer is able to write another exporter in some package and install the package in the system. The `extension_manager` will be able to access the provided functions in the package if they have been declared in the entry points of the `setup.py` script of the named package.

13.2 Exporter

TO DOCUMENT

13.3 Command

TO DOCUMENT

13.4 Importer

TO DOCUMENT

13.5 Explore

TO DOCUMENT

CHAPTER

FOURTEEN

GIT

Papis is conceived to work well with the tool *git*, this would also work with mercurial or subversion.

Here you will find a description of a possible workflow using git with papis. This is not the only workflow, but it is the most obvious.

Let's say you have a library named `books` in the directory `~/Documents/MyNiceBooks`. You could turn the `books` library into a *git* repository, just doing for example

```
papis -l books run git init
```

or just going to the library directory and running the command there:

```
cd ~/Documents/MyNiceBooks
git init
```

Now you can add everything you have in the library with `git add .` if you are in the library's directory or

```
papis -l books git add .
```

if you want to do it using the *papis*' `git` command.

14.1 Interplay with other commands

Some papis commands give you the opportunity of using `git` to manage changes. For instance, if you are adding a new document, you could use the `--commit` flag to also add a commit into your library, so if you do

```
papis add --set author "Pedrito" --set title "Super book" book.pdf --commit
```

then also papis will do an automatic commit for the book, so that you can push your library afterwards to a remote repository.

You can imagine that papis commands like `rename` and `mv` should also offer such functionality, and they indeed do through the `--git` flag. Go to their documentation for more information.

14.2 Updating the library

You can use papis' simple `git` wrapper,

```
papis git pull
```

14.3 Usual workflow

Usually the workflow is like this:

When adding a document that you know for sure you want in your library:

- Add the document and commit it, either by `git add --commit` or committing the document after adding it to the library.
- Pull changes from the remote library, maybe you pushed something at work (reference changes etc..) and you do not have it yet there, you would do something like

  ```
  papis git pull
  ```

- Push what you just added

  ```
  papis git push
  ```

- Review the status of the library

  ```
  papis git status
  ```

When editing a document's info file:

- Edit the file and then take a look at the `diff`

  ```
  papis git diff
  ```

- Add the changes

  ```
  papis git add --all
  ```

- Commit

  ```
  papis git commit
  ```

- Pull/push:

  ```
  papis git pull
  papis git push
  ```

Of course these workflows are just very basic examples. Your optimal workflow could look completely different.

CHAPTER

FIFTEEN

SCIHUB SUPPORT

Papis has a script that uses the scihub platform to download scientific papers. Due to legal caution the script is not included directly as a papis command, and it has its own PyPi repository.

To install it, just type

```
pip3 install papis-scihub
```

Now you can type

```
papis scihub -h
```

and see the help message of the script.

Some usage examples are:

- Download via the doi number:

```
papis scihub 10.1002/andp.19053220607 \\
  add -d einstein_papers --folder-name photon_definition
```

- Download via a url that contains the doi number in the format `.*/doi/<doinumber>`

```
papis scihub http://physicstoday.scitation.org/doi/10.1063/1.881498 \\
  add --folder-name important_paper
```

- Download via the `doi.org` url:

```
papis scihub https://doi.org/10.1016/j.physrep.2016.12.002 add
```

CHAPTER

SIXTEEN

IMPORTING FROM BIBTEX OR ZOTERO

Many users will want to import a library from a bibtex file that another program such as `zotero`, `mendeley` or `jabref` will have exported. These programs usually have a field called `FILE` or `file` that points to a path where the document file can be found.

To import such a library you can use a provided script originally intended for `zotero` which is, however, general enough to work for other programs too.

To install this script you can install the project papis-zotero and follow the instructions therein to import a bibliography file into papis.

CHAPTER

SEVENTEEN

SHELL AUTO-COMPLETION

Papis has a bash auto-completion script that comes installed when you install papis with `pip3`.

It should be installed in a relative path

```
PREFIX/etc/bash_completion.d/papis
```

normally the `PREFIX` part is `/usr/local/`, so you can add the following line to your `~/.bashrc` file

```
source /usr/local/etc/bash_completion.d/papis
```

or get the bash script from here.

17.1 Zsh

There is also a way for `zsh` users to auto-complete. Either downloading the script here. or adding the following line int the `.zshrc` configuration file

```
eval "$(_PAPIS_COMPLETE=source_zsh papis)"
```

CHAPTER

EIGHTEEN

FAQ

Here are some problems that users have come across often:

- When I remove a folder manually in a library or I synchronize the library manually, I do not see the new papers in the library. **Answer**: You probably need to update the cache because papis did not know anything about your changes in the library since you did it by yourself.

  ```
  papis --clear-cache
  ```

  will do.

For more information you can also check the github faq link.

CHAPTER

NINETEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

B

C

D

E

F

G

J

M

O

P

Q

R

S

Y