A tutorial for the knowledge package

Enthusiastic users of the knowledge package

June 30, 2022

Abstract

This is a tutorial on how to to use the knowledge package, together with knowledge-clustering. It shows the basic features of the package, namely how to introduce internal and external hyperlinks on text and math commands, as well as more advanced features. It also contains a guide on how to install and use knowledge-clustering, a "nifty software tool" that aims to ease the use of knowledge.

Contents

Acknowledgments We would like to thank Théo Matricon for helpful feedback on a previous version of this document.

1 Introduction

1.1 The knowledge package

The package knowledge is a package for LaTeX that helps associating information to terms. It can be used for:

- managing external urls, for instance separating the file containing the addresses from their use;
- managing internal references's such as linking every use of a concept to the place of its introduction (in particular avoiding the use of labels);
- managing the index in a centralized way;
- replacing some macros.

Primarily, the goal of knowledge is to produce of scientific documents (the longer, the more interesting, such as a thesis or a book) in order to improve their readability on electronic devices. Ultimately, the goal is to produce documents that are more semantic-aware.

1.2 This tutorial

This tutorial starts with a description of the **basic features of** knowledge in Section 2, followed by an introduction on how to use knowledge-clustering, which is a **command-line tool designed to greatly speed up the process of writing a document** with knowledge in Section 3. Finally, Section 4 deals with more **advanced features** of the knowledge package.

Throughout this document, we will refer to the knowledge documentation. It can be accessed locally by typing `texdoc knowledge` in a prompt or online.

2 Basic features

Try compiling this document (two compilation phases to have proper links) using `pdflatex` and see how some notions are hyperlinked to their introduction point (some viewers make it more obvious than others by displaying a preview of the target of a link inside a document).

2.1 Using knowledge in your LaTeX document

To use knowledge in your LaTeX document, write in the preamble:

```
\usepackage[breaklinks]{hyperref}
\usepackage{xcolor}

\usepackage{knowledge}

\knowledgeconfigure{notion, quotation}
```

By default, `knowledge` is loaded in *composition mode*, which renders links and warnings. The document can be switched to *paper mode*, which is made for printing (links still exist but are displayed in black) or *electronic mode* (links are colored, warnings and anchor points are hidden), by writing

- `\usepackage[paper]{knowledge}` or
- `\usepackage[electronic]{knowledge}`.

2.2 Aesthetical changes and external links

Knowledges are the key concept in the `knowledge` package. Essentially, a knowledge corresponds to a concept used in the document. To invoke a knowledge named "tomato", one simply has to write `\kl{tomato}` (or simply `"tomato"` if the 'quotation' configuration is enabled) in their document. At compilation, this will print the text "tomato" and apply (aesthetical or semantical) changes that are associated with the knowledge "tomato".

To specify what modifications should be performed on a knowledge, you must define it, either in the beginning of your document or in an external file (in `notions.tex` in this example) included in your preamble. The basic syntax to do so is:

```
\knowledge{}
 | tomato
```

Directives can be written between the pair of brackets. A complete list of directives can be found in §5.3 of the `knowledge` documentation. Common directives include:

- `url=<LINK>` to add an external hyperlink;
- `color=<COLOR>` to change the color of the knowledge;
- `italic` and `up` to force/unforce italic;
- `boldface` and `md` to force/unforce boldface;
- `smallcaps` to force small capitals;
- `underline` to underline;
- `lowercase` and `uppercase` to render the text in lowercase or uppercase;
- `typewriter` to render the text in typewriter;

- `text=<TEXT>` to change the text that is displayed.

You will often want to define synonyms, i.e. to have multiple names associated to a single knowledge: for instance you might want "tomatoes", "Tomato" and "Tomatoes" to all refer to the same knowledge as "tomato". This can be achieved by defining each synonym on a new line, precedeed by a pipe. For example:

```
\knowledge{url={https://en.wikipedia.org/wiki/Tomato},
    color=olive, boldface}
 | tomato
 | tomatoes
 | Tomato
 | Tomatoes
```

will produce the following result when one writes `\kl{Tomatoes}` or `"Tomatoes"`:

Tomatoes

namely it will write the text "Tomatoes" in bold, olive, and insert a link to the Wikipedia page named "Tomato".

2.3 Internal hyperlinks: the `notion` directive

The *`notion`* directive allows you to easily introduce internal hyperlinks. Say that you have defined a knowledge:

```
\knowledge{notion, <OTHER_DIRECTIVES>}
 | name
 | synonym
```

By writing `\intro{name}` (or `\intro{synonym}`, or `""name""`, or `""synonym""`) you will *introduce* your knowledge. Then, whenever you will write `\kl{name}` (or `\kl{synonym}`, or `"name"`, or `"synonym"`) `knowledge` will add an internal hyperlink to the place where your `notion` was introduced. The default behaviour[1] is to add a link to the beginning of the section in which the `notion` was introduced. Since this is very often unsatisfying, the command `\AP` allows you to define custom *anchor points*, depicted as small red corners in the left margin of your document when you are in composition mode[2]. Internal hyperlinks will refer to the last anchor point preceding the introduction of your `notion`.

By default, `notions` appear in blue, and introduction of `notions` appear in dark blue and italics. Note that a single `notion` should only be introduced once—even if you have synonyms. Should you want to reintroduce an already introduced `notion`, you can use the `\reintro{...}` command.

[1] Inherited from `hyperref`.

[2] This document was compiled in electronic mode so the anchor points are not shown. However, you can observe anchor points in the minimal example.

2.4 Scopes and extended syntax

Sometimes the same piece of text can refer to different concepts: for example, in this document, "knowledge" refers both to the `knowledge` package and to the concept of knowledges. In this case, *scopes* allow you to distinguish these concepts, by defining the two knowledges:

```
\knowledge{url={https://ctan.org/pkg/knowledge}, typewriter}
 | knowledge@package

\knowledge{notion}
 | knowledge@concept
```

To invoke one or the other, you can write:

```
"knowledge@@scope"
    or
\kl(scope){knowledge}
```

where `scope` is either `package` or `concept`. More informations on scopes can be found in §3.5 of the documentation.

Finally, if you want to display some "text" that behaves like some knowledge named "name", you can write:

```
"text@name"
    or
\kl[name]{text}
```

This is useful when you do not want "text" to be a synonym of "name" throughout the paper but only locally. For instance:

```
(...) "These vegetables@tomato" are (...)
```

produces:

(...) **These vegetables** are (...)

namely the style of the knowledge "tomato" is applied to the string "These vegetables".

2.5 Mathematical commands

The previous sections can mostly be applied to mathematical commands:

```
$\kl[tomato]{\Pi^P_2}$
```

will produce $\mathbf{\Pi}_{\mathbf{2}}^{\mathbf{P}}$. However, as a rule of thumb, this should be avoided as there is a more elegant syntax for knowledgyfied mathematical commands. It is recommended to use semantic macros instead of syntactic ones: for example, instead of defining a macro `\Ac` that displays $\mathcal{A}$, define `\automata` or `\algebra`.

The basic syntax to define a new mathematical command is:

```
\knowledgenewrobustcmd<COMMAND_NAME>{\cmdkl{
    <YOUR_MACRO>
}}
```

For example:

```
\knowledgenewrobustcmd\automata{\cmdkl{
    \mathcal{A}
}}
```

defines a macro named `\automata` that prints an '$\mathcal{A}$' and defines a `notion` named `\automata`. Using the command `\automata` (e.g: $\mathcal{A}$) will result in `knowledge` automatically inserting a link to the last anchor point preceding the introduction of the `notion` `\automata`. This notion can be introduced by writing:

```
\intro*\automata
```

which produces the following result: $\mathcal{A}$.

The `\cmdkl` command allows you to control which part of the macro will be knowledgyfied/clickable. For instance, if you define the macro:

```
\knowledgenewrobustcmd\interval[2]{
    \cmdkl{[} #1, #2 \cmdkl{]}
}
```

then `$\interval{a}{b}$` will produce $[a, b]$: only the two brackets will be clickable.

3 Knowledge-Clustering

3.1 Goal

`Knowledge-clustering` is a command-line tool that aims to automate part of the process of writing a document with `knowledge`. As of today, `knowledge-clustering` has two main features:

- the clustering feature, which automates the definitions of synonyms. For example, if at some point you already defined the knowledge `tomato` and write in your document `"Tomatoes"` then, at compilation, LaTeX will rightfully produce a warning, saying that the knowledge `Tomatoes` is undefined. At this point, you should run `knowledge-clustering`, which will suggest to you to define `Tomatoes` as a synonym of `tomato`.
- the add quotes feature, that can be used at the very end of your writing process, to check if every piece of text that is defined as a knowledge is surronded by quotes. For example, this feature would suggest to replace the string `Let $x$ be a tomato such that (...)` in you `.tex` file by `Let $x$ be a "tomato" such that (...)`.

3.2 Installation

To install knowledge-clustering, you need to have a machine with python3 and pip3. To install, or upgrade, knowledge-clustering, you can simply run

```
pip3 install --upgrade knowledge-clustering
```

in your shell. Then, you should run

```
knowledge init
```

to download some data (roughly 35Mb), used by the clustering algorithm[3].

Autocomplete If the autocomplete of the command knowledge does not work, you can follow the following procedure: if you are using zsh (resp. bash), then add

```
eval "`pip completion --zsh`"
```

or

```
eval "`pip completion --bash`"
```

in your .zshrc file (resp. .bashrc). For the change to take effect, you either need to launch a new terminal or run source ~/.zshrc (resp. source ~/.bashrc).

3.3 Clustering knowledges

3.3.1 Basic use

The *clustering* feature is meant to be used when you are writing your LaTeX document with knowledge. Maintaining the list of all the knowledges you are using can be burdersome, so usually you will write your LaTeX code and use, in this code, some knowledges that are yet to be defined.

For example, Figure 1 illustrates the following situation: in the file containing all your defined knowledges, you have four knowledges, one of which is "word". In your main .tex file—which is not reproduced here—, you used some undefined knowledges, such as "words" and "semigroup". At compilation, LaTeX will produce a warning, saying that you have undefined knowledges and write in a .diagnose file a list of these undefined knowledges.

At this point, you have two options: you can either define every undefined knowledge, by hand, and say that "words" is a synonym of "word" while "semigroup" is a new knowledge. Or, you can use the clustering feature of knowledge-clustering: feed it both files (the file small.tex containing the defined knowledges and the small.diagnose file containing the undefined knowledges) by running the command

```
knowledge cluster -k small.tex -d small.diagnose
```

[3] This downloads some data used by NLTK, a natural language package used by knowledge.

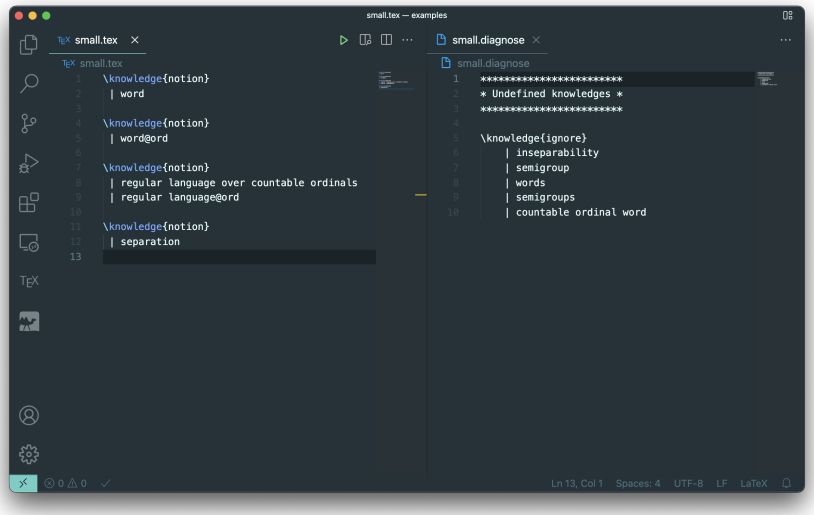

Figure 1: Content of the file containing the defined knowledges (left-hand side) and of the `.diagnose` file produced by LaTeX at compilation (right-hand side), before running `knowledge-clustering`.

which will write suggestions in your file `small.tex`, as depicted in Figure 2. These suggestions take the form of comments: if you agree with the suggestion, you can just uncomment the line and otherwise, you should move it, by hand.

3.3.2 Advanced features

To display the help, you can run `knowledge cluster --help`.

Language The clustering algorithm relies on natural language processing and is language-specific. As of today, only two languages are supported: english (which is the default language) and french. To use `knowledge-clustering` on a document written in french, simply add `-l fr` or `--lang fr` at the end of your command.

Scopes In the example of Figures 1 and 2 you can see that `knowledge-clustering` inferred that the scope "ord" meant "countable ordinals"[4]. If you want the list of scopes it saw and their inferred meaning, you can use the `-S` or `--scope` option. For instance, running:

```
knowledge cluster -k small.tex -d small.diagnose --scope
```

[4]This was inferred by reading the `small.tex` file and was used to cluster the knowledge "countable ordinal word" with "word@ord".

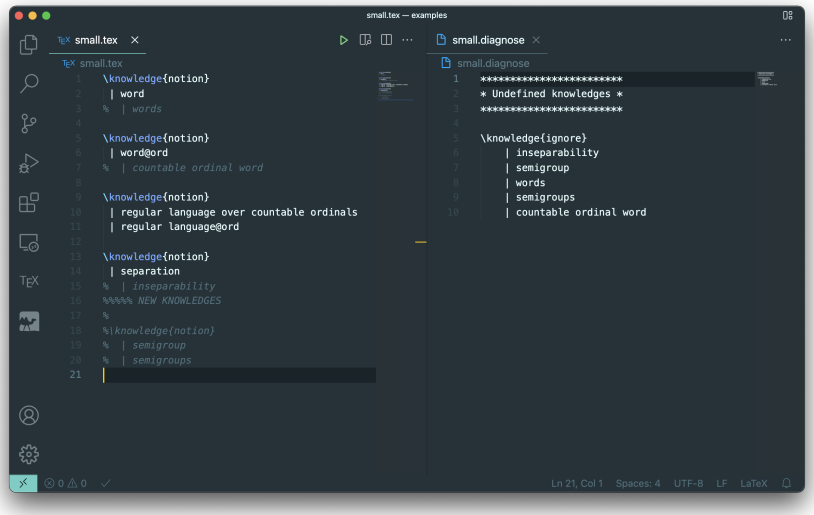

Figure 2: After running `knowledge-clustering`, the `small.diagnose` file is left unchanged, while `small.tex` now contains suggestions of how to define the new knowledges.

will print in your prompt

```
Defined scopes:
@ord : [['ordinals', 'countable'], ['ord']]
```

Config files *Configuration files* are used to specify the list of prefix such that two words that differ on that prefix should be clustered together (suffix are treated using natural language processing). You can define a custom configuration file with your own list of prefixex: it should start with the lines

```
[DEFAULT]
PREFIXES_SIMILAR=
```

and then, on each line, contain a single prefix, preceded by a tabulation, and potentially followed by a comment (which should start with a hash). The default configuration files can be found in the `knowledge-clustering` installation folder under the name

`knowledge_clustering/data/english.ini` and `knowledge_clustering/data/french.ini`.

For instance, the file for english is:

```
[DEFAULT]
PREFIXES_SIMILAR=
```

```
# Empty string
- # ignore dashes
a # (a)chromatic
il
im
in # (in)separable
ir
un # (un)ambiguous
```

This custom configuration file can be given to `knowledge-clustering` using the option `-c` or `--config-file`.

3.4 Forgotten quotes

The add quotes feature is meant to be used when your document is (nearly) finished and you want to check that you have not forgotten quotes symbols (or a `kl{}` command) before and after defined knowledges.

The basic syntax is the following:

```
knowledge addquotes -t <TEX_FILE> -k <NOTION_FILE>
```

where:

- `<TEX_FILE>` is the `.tex` file containing your LaTeX document;
- `<NOTION_FILE>` is the file containing the knowledges you have defined.

Then, your prompt will display something like

```
Found a match for ‘blabla‘ at line 41.
Add quotes? [y/n]
```

Depending on your answer, the add quotes feature will add a quote `"` before, and a quote `"` after the piece of text “blabla” found at line 41.

If you want more precision, you can also print the columns (a tabulation counting as 4 columns) using the option `-c` or `--column`.

```
Found a match for ‘blabla‘ between line 41, column 13 and line
41, column 19.
Add quotes? [y/n]
```

Finally, the option `-F` or `--force` adds quote symbols around every match, without asking your consent first. It is **highly discouraged** to use this option.

3.5 Contributing to `knowledge-clustering`

If you have bugs to report or suggestions, you can submit a new issue, or a new pull request on github, or get in touch by email with one of the maintainers.

If you want `knowledge-clustering` to support a language that is not yet supported, provided that this language belongs to the following list:

- arabic,
- dutch,
- english (already supported),
- finnish,
- french (already supported),
- german,
- hungarian,
- italian,
- norwegian,
- portuguese,
- romanian,
- russian,
- spanish,
- swedish,

the modification should be quite quick. Essentially, it amounts to writing a configuration file for this language, and changing a few lines in this file[5] `app.py`. You can do it yourself and create a new pull request, or get in touch with us.

4 Advanced features

This section is dedicated on more advanced features of `knowledge`.

4.1 Changing the default colors

By default, `knowledge` will print notions in blue and introductions of notions in dark blue if the document is compiled in composition mode or electronic mode, but the external hyperlinks will still be printed in black. Moreover, if it is in composition mode, undefined knowledges will appear in yellow, and anchor points in red.

Unfortunately, the default colors are not maximizing the aesthetical potential of your LaTeX document. Using `\knowledgestyle` (§3.3.4 of the documentation), you can customise these colors. Moreover, `\hypersetup` from the `hyperref` package lets you change the colors of external hyperlinks. Finaly, using

```
\knowledgeconfigure{anchor point color={...}}
```

lets detailed-oriented users change the color of anchors points. However, you want some of these changes to only take effect when you are in paper mode, electronic mode or composition mode[6]: this can be achieved using the commands:

```
\IfKnowledgePaperModeTF{...}{...}
\IfKnowledgeCompositionModeTF{...}{...}
```

For example, this document was produced using the following configuration:

[5] More precisely, you should change the constants `CONFIG_FILENAME`, `NLTK_LANG`, the definition of `knowledge init`, and the possible choices for the `-lang` option of `knowledge cluster`.

[6] Fun fact: the paper mode is used by LIPIcs when compiling proceedings. Should you decide to color your links even in paper mode, you might make Dagstuhl annoyed.

```
    \definecolor{Dark Ruby Red}{HTML}{7c1b1e}
    \definecolor{Dark Blue Sapphire}{HTML}{004c5c}
    \definecolor{Dark Gamboge}{HTML}{be7c00}

    \IfKnowledgePaperModeTF{
        %
    }{
        % If we are NOT in paper mode (i.e. in composition mode or
electronic mode)
        \knowledgestyle{intro notion}{color={Dark Ruby Red}, italic}
        \knowledgestyle{notion}{color={Dark Blue Sapphire}}
        \hypersetup{
            colorlinks=true,
            breaklinks=true,
            linkcolor={Dark Blue Sapphire}, % Links to sections, pages,
etc.
            citecolor={Dark Blue Sapphire}, % Links to bibliography
            filecolor={Dark Blue Sapphire}, % Links to local file
            urlcolor={Dark Blue Sapphire},
        }
        \IfKnowledgeElectronicModeTF{
            %
        }{
            % If we are in composition mode, highlight unknown stuff
(in yellow) and display the anchor point.
            \knowledgeconfigure{anchor point color={Dark Ruby Red},
anchor point shape=corner}
            \knowledgestyle{intro unknown}{color={Dark Gamboge}, italic}
            \knowledgestyle{intro unknown cont}{color={Dark Gamboge},
italic}
            \knowledgestyle{kl unknown}{color={Dark Gamboge}}
            \knowledgestyle{kl unknown cont}{color={Dark Gamboge}}
        }
    }
```

This produces the following result: in composition mode, every link (both internal and external) appears in a nice shade of blue, called "Midnight Green Eagle", definitions and anchor points are displayed in dark red ("Ruby Red"), and unknown knowledges are displayed in dark yellow ("Gamboge"). In electronic mode, everything works the same except that anchor points are not displayed and unknown knowledges are not highlighted. Finally, in paper mode, everything is printed in black.

4.2 Weird spacing in math commands

Say you want to define a binary relation: without `knowledge` you would write something like:

```
\newrobustcmd{\myrelation}{%
    \mathrel{\mathcal{R}}%
}
```

which will have the desired behaviour: writting `$x \myrelation y$` will produce $x \mathcal{R} y$ (i.e. the relation symbol is precedeed and followed by spaces), and writting `let $\myrelation$ be a relation` produces "let $\mathcal{R}$ be a relation": everything works as expected. Now try to knowledgify this command, by defining it like:

```
\knowledgenewrobustcmd{\myklrelation}{%
    \mathrel{\cmdkl{\mathcal{R}}}%
}
```

which will produce $x \mathcal{R} y$ and "let $\mathcal{R}$ be a relation"[7]. The command does not behave as expected in the second case! This is because of the definition of `\knowledgenewrobustcmd` (see §3.9.1 of the documentation), which is essentially equivalent to:

```
\newrobustcmd\myklrelation{%
    \withkl{\kl[\myklrelation]}{%
        \mathrel{\cmdkl{\mathcal{R}}}
    }
}
\knowledge{\myklrelation}{notion}
```

To obtain the desired behaviour, one has to change the position of the `\mathrel` command like so:

```
\newrobustcmd\mynewklrelation{%
    \mathrel{%
        \withkl{\kl[\mynewklrelation]}{%
            \cmdkl{\mathcal{R}}
        }%
    }%
}
\knowledge{\mynewklrelation}{notion}
```

which produces the following result: $x \mathcal{R} y$ and "let $\mathcal{R}$ be a relation". More details can be found in §3.9.2 of the documentation.

4.3 Other functionalities

This tutorial is far from being exhaustive: you can take a look at the documentation to see all the possible wonders `knowledge` has to offer, such as handling indexes, knowledgifying already-defined commands, defining command behaving diffently depending on whether they are used in text or math mode, etc.

[7] You can observe here a fun phenomenon: if a knowledge is defined but not introduced, it will still be displayed in blue—but it will not be clickable.