A tutorial for the knowledge package

Enthusiastic users of the knowledge package

June 23, 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

1 Introduction

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 the `knowledge` is for the production 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.

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

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 the *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}`, respectively.

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 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. Most basic example 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=purple, 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, purple, and insert a link to the Wikipedia page named "Tomato".

2.2 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 writting `\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. 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 `\reinto{...}` command.

2.3 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

[1] Inherited from `hyperref`.

```
"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".

2.4 Mathematical commands

The previous sections can mostly be applied to mathematical commands: for instance

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

will produce $\mathbf{\Pi_2^P}$. However, as a rule of thumb, this should be avoided as there is a more elegant syntax for knowledgyfied mathematical commands. It is recommanded 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 writting

```
\intro*\automata
```

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

The `\cmdkl` command allows you to control which part of the macro will be knowledgyfied/cliquable. 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 cliquable.

3 Knowledge-Clustering

3.1 Goal

`Knowledge-clustering` is a command-line tool that aims to automate part of the process of writting a document with `knowledge`. As of today, `knowledge-clustering` has to 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 propose to you to define `Tomatoes` as a synonym of `tomato`.
- the add quotes feature, that can be used at the very end of your writting 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, used by the clustering algorithm[2].

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`).

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

3.3 Clustering knowledges

3.3.1 Basic use

The *clustering* feature is meant to be used when you are writing you LaTeX document with `knowledge`. Maintaining the list of all the knowledges you're 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.

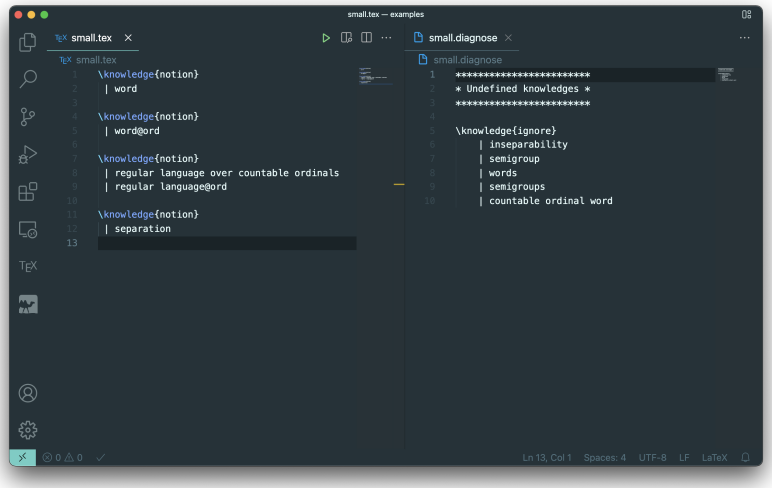

Figure 1: Content of the file containing the defined knowledges (left-hand side) and of the `.diagnose` file produced by LaTeX at compilation (right-hand side), before running `knowledge-clustering`.

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 -n small.tex -d small.diagnose
```

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.

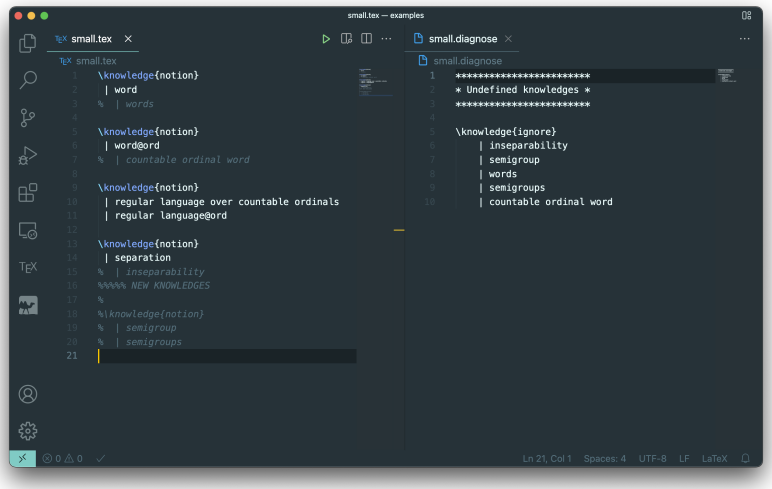

Figure 2: After running `knowledge-clustering`, the `small.diagnose` file is left unchanged, while `small.tex` now contains suggestions of how to define the new knowledges.

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` infered that the scope "ord" meant "countable ordinals"[3]. If you want the list of scopes it saw, and of their infered meaning, you can use the `-S` or `--scope` option. For instance, running

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

will print in your prompt

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

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

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.

4 Advanced features

4.1 Weird spacing for math commands

4.2 Disabling commands

4.3 Changing the default colors