`yacite` tutorial

Motto

> This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.
>
> Bell Laboratories. M. D. McIlroy, E. N. Pinson, and B. A. Tague. "Unix Time-Sharing System Forward". *The Bell System Technical Journal* 1978. **57** (6, part 2). p. 1902.

What is `yacite`

`yacite` is a system commandline UNIX utilities to solve the following tasks:

1. Keeping track of scientific papers written by a single author or a small group.
2. Keeping track of citations of one's papers.
3. Simple filtering, automatic manipulation, tagging etc. of bibliographic records.
4. Rendering subsets of bibliographic data in HTML or other text-based form.

Lesson 1: Creating HTML list of your publications

1. Open a terminal emulator and type the following commands on the shell prompt

   ```
   mkdir yacite_sandbox
   cd yacite_sandbox
   ```

   This creates a new working directory.

2. You will need a BibTeX ('.bib') database file containing your publications. You can download it from a bibliographic database like Scopus of Web of Science or Google Scholar. Save the file (let us call it `mypubs.bib`) in the `yacite_sandbox` directory you created.

3. Go back to the shell prompt (the current directory should be `yacite sandbox` and type the following command

   ```
   bib2yaml mypubs.bib
   ```

This command prints on its standard output a series of records in YAML format, we call this *a YAML stream.* A YAML stream consists of a sequence of records that look like this:

```
authors: ['Jenča, Gejza']
doi: 10.1007/s00500-012-0901-x
endpage: 47
grants: VEGA G-2/0059/12, VEGA G-1/0297/11, APVV-0073-10, APVV-0178-11
issn: 1432-7643
issue: 1
journal: Soft Computing
keywords: Effect algebra; Lattice effect algebra; Compatibility center
publisher: Springer-Verlag
startpage: 45
tags: [zpub, sci, cc, scopus]
title: Congruences generated by ideals of the compatibility center of lattice efect alg
type: article
url: http://dx.doi.org/10.1007/s00500-012-0901-x
volume: 17
year: 2013
```

The records in the YAML stream are separated by triple minus `---`.

Unlike BibTeX, YAML is a popular way to represent data in the current world of computing. Probably in every programming language, there is a library to read and write YAML data. Unlike other serialization formats (like XML), the YAML format was designed to be both human-readable and human-editable. It can be edited using any text editor. 1. What we want to do now is to convert a YAML stream into a HTML document. This is accomplished using a *template*. A HTML template is a HTML file containing special non-HTML control sequences representing variables, loops, conditions etc. `yacite` uses the jinja2 template language.

The templates are kept in a separate directory. Create the directory on the command prompt:

```
mkdir templates
```

To keep things simple, we shall create a very simple template. Open your favorite text editor and type in the following jinja2 code.

```
<html>
<body>
<h1>List of publications</h1>
<ol>
# for rec in records:
<li>
```

```
{{  rec.authors|join(", ") }}
<em>{{rec.title}}</em>,
{{rec.journal}},
<strong>{{rec.volume}}</strong>
({{rec.year}})
{{rec.startpage}}-{{rec.endpage}}
</li>
# endfor
</ol>
</body>
</html>
```

This is pretty much self explanatory. The jinja2 commands start with `#`. The variables are enclosed in double curly brackets. The construction

```
{{  rec.authors|join(", ") }}
```

means that the authors should be separated by a comma and a space. 1. Save the template under some name, let us say `publist.html` in the `templates` directory. 1. Type in the following command on the shell prompt.

```
bib2yaml mypubs.bib | yacite render publist.html
```

This is called a *pipe sequence*. The whole `yacite` system is based on pipe sequences. There are two commands at work here. The `bib2yaml` command produces a YAML stream as its output. The `|` sign (called *pipe*) in the middle connects the output of the `bib2yaml` command and with the input of the other command, `yacite render`. The `yacite render` command reads a YAML stream on its input, fills in the template, and then produces HTML on its output.

There are more things that `yacite render` does (like dealing with citations) but, for the moment, we do not care. 1. Save the resulting HTML into a file using the redirect sign `>` :

```
bib2yaml mypubs.bib | yacite render publist.html > mypubs.html
```

and open the `mypubs.html` in you favourite browser. Enjoy the result. 1. Try to alter the template; for example, change the `<ol>` and `</ol>` tags to `<ul>` and `</ul>`, respectively, and observe the results in `mypubs.html`. Note that you need to click the reload button in your browser to see the updates in a HTML document. 1. If you need to create a MS Word document, there are some ways how to convert a simple HTML document to a `.doc` file. For example, you can try the excellent `pandoc` document conversion utility. In addition, as I was told, MS Word should be capable to load a HTML file and save it as a MS Word file.

Lesson 2: Filtering the YAML stream

1. Try the following command

   ```
   bib2yaml mypubs.bib | yacite filter 'year == 2012'
   ```

   Note that the argument of the `yacite filter` command in enclosed in apostrophes. This tell the shell to pass it to the `yacite` as a single argument. The argument `year == 2012` is a Python expression.

   The result should be a filtered YAML stream that contains only the 2012 publications.

   Explanation: The `yacite filter` command evaluates the `year == 2012` expression in the environment given by the record in stream. If the expression is true, the records is outputted, otherwise it is skipped.

   You can now try create a HTML document using `yacite render`, as in the previous example.

   ```
   bib2yaml mypubs.bib | yacite filter 'year == 2012' | yacite render publist.html > mypub
   ```

2. Alter the argument of `yacite filter` as follows:

   ```
   bib2yaml mypubs.bib | yacite filter '"the" in title' | yacite render publist.html > myp
   ```

   This should create a list of publications containg 'the' as a substring of the title.

3. You can use conjunction (`and`), disjunction (`or`), negation (`not`):

   ```
   ... yacite filter '("Fourier" in title) or "(Laplace" in title)' ...
   ... yacite filter '(year > 2000) and (year <= 2010)' ...
   ... yacite filter 'not ("semigroup" in keywords)' ...
   ```

   The last command is equivalent to much more readable

   ```
   ... yacite filter '"Effect" not in keywords' ...
   ```

   We refer to Python tutorial for more information about Python expressions.

Lesson 3: Persistence: saving the data into database directory

1. In `yacite`, data are saved in plain YAML documents, one record per file. The documents are saved in a database directory, called *datadir*. Currently, datadir simple and allows for editing the data in place using any text editor, since there are no metadata.

   Create the datadir:

   ```
   mkdir pubcit
   ```

   The name of the datadir does not matter at all. 1. The command `yacite merge` reads the YAML stream from its input and saves the individual records in the datadir.

   ```
   bib2yaml mypubs.bib | yacite merge pubcit
   ```

   You see a couple of log messages saying that some new records were created. Go into the `pubcit` directory and look around. There is a couple of files with the name scheme

   `pubcit/${YEAR}/${AUTHOR}${YEAR}${TITLEWORD}.yaml`

   `${AUTHOR}` is the surname of the first author. `${TITLEWORD}` is the first word in the title, unless it is too short, like “a” or “the”. In this case, `${TITLEWORD}` are first two word of the title. The name scheme makes it easy to find individual records.

   When you open one the files you in a text editor can see that in every file a new field `key` was created. The value of `key` is `${AUTHOR}${YEAR}${TITLEWORD}`. It is easy to infer the value of `key` from the basic bibliographic data of the paper. 1. Sometimes, there are multiple records with the same first author, year and the first word of the title. Whenever this happens, a suffix `-1` (or more) is appended to `key` and to the corresponding part of the filename. 1. Try again the command

   ```
   bib2yaml mypubs.bib | yacite merge pubcit
   ```

   Hopefully, nothing happens. For every record on its input, `yacite merge` figured out using a heuristics that the record on its input points (in the real world) to the same article as the one stored in the database. Since every filed value in the input record (except for the `key`, which is ignored in this respect) is the same as the stored one, there nothing to do and the stored record is unchanged.

Lesson 4: Marking the records as your own

1. Let us start with a clean table. Remove the datadir and start with an empty one.

   ```
   rm -rv pubcit
   mkdir pubcit
   ```

2. Try this command

   ```
   bib2yaml mypubs.bib | yacite exec ’myown=True’
   ```

   The result is an altered YAML stream. There is a new field, `myown` with value `true` in every record. Note the difference; `True` is a Python constant, `true` is a YAML constant. Also, there is a single `=`, not `==`; we are assigning a value, not making a comparision.

   Explanation: The `yacite exec` command executes the Python command (in this case, `myown=True`) in the environment given by the record in stream. The transformed records are written to the standard output of `yacite exec` as a YAML stream. 1. Now let the altered YAML stream flow into the `yacite merge` command:

   ```
   bib2yaml mypubs.bib | yacite exec ’myown=True’ | yacite merge pubcit
   ```

   Note that the filenames are different now: every new file is placed into a `myown` subdirectory. This allows one to easily find his/her own records.

Lesson 5: Reading the database directory

1. The command `yacite read` reads all data in the datadir and outputs them as a YAML stream:

   ```
   yacite read pubcit
   ```

   You can render the stream using a template:

   ```
   yacite read pubcit | yacite render publist.html > out.html
   ```

   Try to put some `yacite filter` in the middle of the pipe sequence and look at the result.

Lesson 6: Tagging the records in the database directory

1. There are two ways how to update records in the datadir: either direct editing or via the `yacite merge` command. Direct editing should be clear: just edit the files using any text editor. This is aprropriate for individual changes.

2. Let us supose we that all the data in datadir were created from SCOPUS. We would like to mark them as such. One of the possibilities would be to add a boolean variable `scopus` in every record, as we did with `myown` in one of the above examples.

   ```
   yacite read pubcit | yacite exec 'scopus=True' | yacite merge pubcit
   ```

   However, this is not necessarily a good idea, because, for example, it is not clear from looking at the data which fields are "real data" and which are "custom marks". It is better to equip every record with a list of strings, let us say `tags`. There is a special pair of commands for manipulation with lists of strings, namely `yacite append` and `yacite unappend`. Look at the output of this command:

   ```
   yacite read pubcit | yacite append tags scopus
   ```

   There is `tags: [scopus]` in every record in the resulting YAML stream. Let the stream flow back into the database:

   ```
   yacite read pubcit | yacite append tags scopus | yacite merge pubcit
   ```

   The merge command displays a couple of log messages about creating new `tags` fields for every record in the database. 1. Let us mark the records older than 2005 as old. This can be done using sequence of filtering and marking:

   ```
   yacite read pubcit | yacite filter 'year<2005' | yacite append tags old
   ```

   This works as expected, the resulting YAML stream contains either '[scopus,old]' or '[scopus]' as the value of 'tags', depending of the value of year. However,

   ```
   yacite read pubcit | yacite filter 'year<2005' | yacite append tags old | yacite merge
   ```

   produces a message about "bounced" fields. To see what happened, try the '-v' option (v is for verbose):

```
yacite read pubcit | yacite filter 'year<2005' | yacite append tags old | yacite merge
```

Every changed `tags` field bounced. What happened? The `yacite merge` command recognized that the records in the YAML stream are new versions of the records in the database and that the value of `tags` is different. `yacite merge` is careful, it does not change existing data in the database unless explicily said to do so. And did not say what to do with the changed value of `tags`. What we want to do is to set `tags` to the new value. This command does the trick:

```
yacite read pubcit | yacite filter 'year<2005' | yacite append tags old | yacite merge
```

The `-s tags` (s stands for SET) says that the value of `tags` should be updated to the value obtained from the YAML stream.

3. Let us delete the 'old' tag (it was a silly example anyway).

   ```
   yacite read pubcit | yacite unappend tags old
   ```

   shows a YAML stream without any `old` tag among `tags`. The rest of the task of deleting the `old` tag is left as an exercise to the esteemed reader.

Lesson 7: Merging data from different sources

1. Let us start with a clean database again:

   ```
   rm -r pubcit
   mkdir pubcit
   ```

2. We assume in this lesson that we have two BibTeX files containing bibliography data concerning your own articles, the names of the files are `scopus.bib` and `wos.bib`. Merge the data from the `scopus.bib` into the database, marking them with `scopus` tag and `myown=true` in each record.

   ```
   bib2yaml scopus.bib | yacite append tags scopus | yacite exec 'myown=True' | yacite mer
   ```

3. Take the other file, `wos.bib` and do the same thing with it, use the `-v` (verbose) option with yacite merge.

   ```
   bib2yaml wos.bib | yacite append tags wos | yacite exec 'myown=True' | yacite merge -v
   ```

   What happened? Examine the messages given by `yacite merge`.

(a) If there are some records that are in `wos.bib` and not in `scopus.bib`, they were new so the `yacite merge` did the right thing: it created new records. These are exactly the records that are in `wos.bib` and not in `scopus.bib`. Their value of `tags` is `[wos]`.

(b) For the records that are in both BibTeX files, there were some new fields present in YAML stream. These were appended to the corresponding records.

(c) A lot of fields "bounced". That means that they were present in both records, but their values were different. It is for sure that the field `tags` is amoung the bounced fields, since its value is `[scopus]` in the database and `[wos]` in the YAML stream. It is possible that the other fields differ as well, mostly `title`, `journal`, `authors` and others. Probably the only difference is the capitalization and whether the authors field contains full names.

4. What we need to do now, is to set the value of the field `tags` in a record in the database to `[scopus, wos]` for every record in the database that matched a record in the YAML stream originated; these are the records that are in both `wos.bib` and `scopus.bib`. For each matching pair of records, the record in the YAML stream has `tags: [wos]` and the record in the database has `tags: [scopus]`. So we instruct `yacite merge` to make a union (`-u`) of the value of `tags`:

```
bib2yaml wos.bib | yacite append tags wos | yacite exec 'myown=True' | yacite merge -v
```

Lesson 8: Dealing with bounced fields

1. When merging data from different sources, it is often necessary to deal with the fact that the fields that (ideally) should have the same value are, in fact different. Examples follow.

   - The first name of one of the authors is (not) given.
   - The capitalization in title differs.
   - The capitalization in journal or series name differs.
   - The name of the journal contains a subtitle.

   The `yacite merge` command detects that the fields are different, but it does not do anything about it without an user intervention. 1. Try the following command:

```
bib2yaml wos.bib | yacite append tags wos | yacite exec 'myown=True' | yacite merge -b
```

   You should see a short message about the number of bounced fields/records. Open the file `diff.yaml` in a text editor. For every record that has at least one bounced field, it contains the following data:

- The `key` field of the record in the database. This allows to identify the database record unambigously.
- The names and values from the input YAML stream that bounced.
- The field names and values of the bounced fields that are present in the database are given in a remark. This allows you to compare the old value with the new one to decide which one is better.

So now you could open the corresponding database file in a text editor and copy/paste the values from `diff.yaml`, right? Wrong!

The key point is that the file `diff.yaml` now contains a *mergeable YAML stream!*. So the following command is legal, but has no effect on the database:

```
yacite merge pubcit < diff.yaml
```

If you edit the `diff.yaml` so that it contains only the fields that are "better" in `diff.yaml` than in the database, you can tell `yacite merge` to rewrite the field with the new value. This is the right way to deal with the problem of bounced fields.

For example:

(a) Pick some important field, let us say `authors`

(b) Delete every line `authors:...` in `diff.yaml` where the value in `diff.yaml` is "worse" than the one in the database.

(c) Use the following command (note that the output goes to a different file than `diff.yaml`)

```
yacite merge -b -s authors pubcit < diff.yaml > diff2.yaml
```

The `-s authors` means: SET `authors` to the new value. 1. Look at the message: the value of `authors` was updated in the database. The `diff2.yaml` file contains remaining bounced fields. 1. Repeat for another field you consider important.

Of course, you can edit all the fields at once and then merge them with a single `yacite merge` command; there can be multiple `-s` options when doing a merge.

Lesson 8: Adding citations – per cited article

1. Let us assume you already managed to register all of your own papers in the database. Now go to the citation database, for example SCOPUS and export all citations of *one* of your papers to a file, let us say `citations.bib`.

2. Let us say that the `key` of your paper is `yourname2012aword`. Run the following command to merge the citations.

   ```
   bib2yaml citations.bib | yacite append cites yourname2012aword | yacite append tags sco
   ```

3. Repeat for all of your papers. Maybe you have to deal with some bounces, especially when you use multiple citation databases.

4. This workflow is more effective for creating the initial database of citations,simply because the number of cited papers is probably smaller than the number of citing papers.

Lesson 9: Adding citations – per citing article

1. There is another type of workflow for adding citations, probably more effective for updating the database with new citing papers.

2. Export the bibliographic data of the new citing papers to a file, say `newcites_scopus.bib`.

3. Convert the file to yaml:

   ```
   bib2yaml newcites_scopus.bib | yacite append tags scopus |
   yacite exec 'cites=[]' > newcites_scopus.yaml
   ```

   (a) Edit the `newcites_scopus.yaml` file. Fill the value of the cites field with corresponding keys of your articles. Then, merge the edited file: yacite merge -u tags -u cites < newcites_scopus.yaml

Lesson 10: Rendering citation lists

1. We already know how to represent citations. How do we get the information in a presentable form? Using a jinja2 template, of course. This is a snippet of the HTML template I use, I call it `pubcitlist.html`. Recall that default location of templates is the `template` subdirectory.

   ```
   <html>
   <body>
   # for rec in records:
   # if rec.myown
     # if rec.citedby:
     <h3>Paper:</h3>
     <p>
     {{  rec.authors|join(", ") }}:
     <em>{{rec.title}}</em>,
   ```

```
  {{rec.journal}},
  <strong>{{rec.volume}}</strong>
  ({{rec.year}})
  {{rec.startpage}}-{{rec.endpage}}
  </p>
  <h3>
  Cited in:
  </h3>
  <ol>
  # for rec_cited in rec.citedby:
    <li>
    {{  rec_cited.authors|join(", ") }}:
    <em>{{rec_cited.title}}</em>,
    # if rec_cited.journal:
    {{rec_cited.journal}},
    # endif
    # if rec_cited.series:
    {{rec_cited.series}},
    # endif
    <strong>{{rec_cited.volume}}</strong>
    ({{rec_cited.year}})
    {{rec_cited.startpage}}-{{rec_cited.endpage}}
    # if "as-preprint" in rec_cited.tags:
    (cited as preprint)
    # endif
    </li>
  # endfor
  </ol>
  <hr>
  #endif
# endif
# endfor
</body>
</html>
```

There are two cycles in the template: the outer cycle runs through all records, but it considers only those that have `myown = true` and are cited at least once. For each such record `rec`, the inner cycle runs through all records in `rec.citedby`; this is a list of records that is *dynamically generated* by the `yacite render` command.

2. To render a subset of the citations, all you have to do is to use a `yacite filter`. A couple of examples follow.

   Render all citations of all my papers in the database:

```
yacite read pubcit | yacite render pubcitlist.html > out.html
```

Render all citations of my papers from 2012 in the database

```
yacite read pubcit | yacite filter --myown 'year==2012' |
yacite render pubcitlist.html
```

Render all citations from 2012 of all my papers in the database

```
yacite read pubcit | yacite filter --notmyown 'year==2012' |
yacite render pubcitlist.html
```

So `--myown` restricts the filtering expression only to records with `myown = true`, the others (with `myown = false` of without `myown`) pass through the `yacite filter` unconditionally. Similarly for `--notmyown`.

Lesson 11: What now?

1. Well, you should read the reference, there is a couple of commands and a lot of options that were not described in this tutorial. Figure out the workflow that suits you best.

2. You should read the template documentation in the jinja2 home page: [http://jinja.pocoo.org/docs/] .

   Jinja2 can do a *lot* for you. 1. Maybe you need a little bit of Python – to write the more elaborated filters and execs. Python docs are at [http://docs.python.org/2/].