The DuckBoat — News from TEX.SE: Processing text files to get LATEX tables

Herr Professor Paulinho van Duck

Abstract

In the first part of this installment, Prof. van Duck will talk about the distribution of upvotes by topics on TEX.SE. In the following Quack Guide, you will find out how to easily process text files to create professional LATEX tables.

1 Quack chat

Hi, (LA)TEX friends!

An amazing event happened on October 20th, 2018, in Rome, Italy: my first talk! To tell the truth, since I was too shy to speak, my friend Carla helped me with my presentation. It was a very exciting experience!

I would like to thank all the GUIT friends who were present at the meeting and listened with patience.[1]

For those who love little animal icons, an awesome piece of news is that the gorgeous

package is now on CTAN.

Some duck fans use it to create the welcome party video[2] for Barbara Beeton: she retired in February, so she will have much time to spend on TEX.SE, all the Ti*k*Zlings are waiting for her!

Last but not least, on December, 17th, *The New York Times* dedicated an article to our Jedi Master, Prof. Donald E. Knuth, "The Yoda of Silicon Valley".[3] Unmissable reading for any TEX fan, quack!

TEX.SE friends are always very attentive in finding typos or inaccuracies in my articles.

The user 'TeXnician' pointed out there is a `text dept` instead of `text depth` in Figure 2 of *The Morse code of TikZ* in *TUGboat* 39:1.

About *Formatting posts* in *TUGboat* 39:3, marmot remarked that, on a Mac, shortcuts are usually done with the command key, even though Ctrl also works.

I would like to thank them both.

This time I will show you how to process text files to automatically obtain professional LATEX tables. As we will see, it is straightforward and convenient.

But let us talk about the upvoting distribution by topics on TEX.SE, first.

2 "Cinderella" topics

Not all TEX.SE tags have the same level of popularity.

Since I was curious to know which were the "Cinderella" topics, that is, less valued but not less worthy, I asked my friend Carla to post a question on Meta[4] for me.

We got a gorgeous answer by moewe, with a very detailed and accurate analysis of the vote distribution. His results are somehow surprising, quack!

First, there is a general feeling that Ti*k*Z related posts receive, on average, many more upvotes than others. Looking at the data, this is not entirely true.

Taking into account only the tags with more than 5,000 answers, there are some which perform better than Ti*k*Z, such as `symbols`, `macros`, and `math-mode`. Extending to tags with more than 2,000 answers, the absolute top topic is `tex-core`.

Looking at the "Cinderella" ones, meanwhile, moewe found out that `table-of-contents`, `floats`, and `tables`, among the most frequent tags, perform worse than other topics.

This is quite unexplainable because these are the first things beginners found "strange", compared with ordinary word processors. Maybe the posts are trivial or focused on such peculiar problems that they do not matter for other users.

Extending the sample, also `bibliographies` and LyX can keep the seven dwarfs company.

The low reputation average of the latter is not a surprise. Many TEXnicians do not like the WYSIWYG/M (What You See Is What You Get/Mean) philosophy adopted by this tool.

I do not think the idea is wrong *a priori*. I used this tool in the past; I gave up only because my LyX files where so full of ERT (Evil Red Text) that it was more convenient to write them in LATEX directly.

As for `bibliographies` (especially BIBLATEX), maybe the few upvotes are due to the peculiarity of the questions, which specifically concern the problem of a single OP and are often not useful for others. Moreover, we have few BIBLATEXperts (among which

[1] For more information: `https://www.guitex.org/home/guit-meeting-2018`.

[2] `https://vimeo.com/315852862`.

[3] `https://www.nytimes.com/2018/12/17/science/donald-knuth-computers-algorithms-programming.html`.

[4] `https://tex.meta.stackexchange.com/questions/7977/poll-which-are-the-cinderella-topics`.

moewe is undoubtedly the top one) who can fully understand and appreciate these answers.

One of the "Cinderella" tags is `csvsimple`. It has only 2.8 votes per answer, on average, and more than 8% of answers with no votes at all. Since I think this package is useful and I would like to make it more popular, I have dedicated the current Quack Guide to it.

3 Quack Guide No. 4 How to easily process text files to get LaTeX tables

We all know we can get beautiful tables with LaTeX, but typing them can be boring and error-prone.

We often have our data processed by another tool which yields a file in some text-based format (`.csv`, `.dat`, `.tex` or similar), and we only need to transform it into a LaTeX table.

There are some packages for this purpose, for example, `pgfplotstable` and `datatool`. I will show you `csvsimple`, which is the simplest one (that is the reason for its name, quack!) but sufficient for basic usage.

Suppose we have the CSV file `test.csv`, which contains the following data (any resemblance to real persons/ducks is purely coincidental):

test.csv

```
Name,Surname,Height,Gender
Paulinho,van Duck,.4,M
Paulette,de la Quack,.35,F
Enrichetta,Pescatore,1.80,
Henry,Gregory,,M
```

Pay attention to the missing data: the comma (or, in general, the separator) is needed, see, for example Enrichetta's "Gender" or Henry's "Height".

With this simple code:

```
\documentclass{article}
\usepackage{csvsimple}
\usepackage{booktabs}
\begin{document}
\csvautobooktabular{test.csv}
\end{document}
```

we can already get a passable result, as shown in Table 1.

I used `\csvautobooktabular` because my table fits on one page; for multiple-page tables, there is `\csvautobooklongtable`.

Table 1: Result of `\csvautobooktabular` applied to our comma-separated test file with header line.

Name	Surname	Height	Gender
Paulinho	van Duck	.4	M
Paulette	de la Quack	.35	F
Enrichetta	Pescatore	1.80	
Henry	Gregory		M

As you may note, `csvsimple` does not itself load `booktabs` nor `longtable`; hence you have to load them separately.

The macros that do not need `booktabs`, that is `\csvautotabular` and `\csvautolongtable`, produce tables with vertical lines. However, since TeXnicians love vertical lines in tables as much as chairs covered with cactus, I recommend not using them, quack!

Of course, we can improve our table.

For example, suppose we want to correctly align the numeric data and change their heading, adding the unit of measure, and also centering the "Gender" column and swapping it with column "Height".

All that can be done by putting a `\csvreader` within a `tabular` environment:

```
\documentclass{article}
\usepackage{csvsimple}
\usepackage{booktabs}
\usepackage{siunitx}
\usepackage{makecell}

\begin{document}
\begin{tabular}{
  lcS[table-format=1.2,round-mode=places]}
  \toprule
  Person/Duck & Gender &
    {\makecell{Height\\ (\si{\metre})}}\\
  \midrule
  \csvreader[head to column names,
    late after line=\\]{test.csv}{}%
    {\Name\ \Surname & \Gender & \Height}
  \bottomrule
\end{tabular}
\end{document}
```

or, directly, with the appropriate options in the `\csvreader` command (in the following I will show only the `\csvreader` commands; the rest of the code is the same as before):

Table 2: A table created with `\csvreader` applied to our comma-separated test file with header line.

Person/Duck	Gender	Height (m)
Paulinho van Duck	M	0.40
Paulette de la Quack	F	0.35
Enrichetta Pescatore		1.80
Henry Gregory	M	

```
\csvreader[
  tabular={
    lcS[table-format=1.2,round-mode=places]},
  table head={\toprule
    Person/Duck & Gender &
      {\makecell{Height\\ (\si{\metre})}}\\
    \midrule},
  head to column names,
  late after last line=\\\bottomrule,
  ]{test.csv}{}%
  {\Name\ \Surname & \Gender & \Height}
```

Please note that I have also created a unique column with name and surname, simply by using: `\Name\ \Surname`

The output of the two previous examples is the same, shown in Table 2.

The general syntax of the command is

`\csvreader[`⟨*options*⟩`]{`⟨*file name*⟩`}%`
`{`⟨*assignments*⟩`}{`⟨*command list*⟩`}`

where ⟨*file name*⟩ is the name of your text file and ⟨*command list*⟩ is executed for every line.

The ⟨*options*⟩ can provide instructions for the table formatting, as follows.

With `tabular=`⟨*table format*⟩ you can specify the column types you prefer.

In `table head=`⟨*code*⟩ you can insert the code for the table headings.

`late after line=`⟨*code*⟩ is the code to be executed after processing a line of the input file;[5] if the option `tabular` is present, it is set to `\\` automatically. Analogously, `late after last line=`⟨*code*⟩ is executed after processing the last line of the file.

When `head to column names=true|false` is set to `true` (the default), the entries of the header line of the input file are used automatically as macro names for the columns.

This option cannot be used if the header entries contains spaces, numbers or special characters, because only letters can be used in LaTeX macro names. If this is the case, you can set `head to column names=false` and reference the columns with the `csvreader` macros: `\csvcoli` (`coli` means the first column; roman numerals are used since arabic numerals cannot appear in LaTeX commands), `\csvcolii`, `\csvcoliii`, and so on:

```
\csvreader[
  tabular={
    lcS[table-format=1.2,round-mode=places]},
  table head={\toprule
    Person/Duck & Gender &
      {\makecell{Height\\ (\si{\metre})}}\\
    \midrule},
  head to column names=false,
  late after last line=\\\bottomrule,
  ]{test.csv}{}%
  {\csvcoli\ \csvcolii & \csvcoliv &
    \csvcoliii}
```

Alternatively, you can take advantage of the ⟨*assignments*⟩ parameter, giving a customized macro name to the column entry, with ⟨*name*⟩=⟨*macro*⟩, where ⟨*name*⟩ is the arabic number of the column (or the entry from the header, if you want to use another macro to identify the column):

```
\csvreader[
  tabular={
    lcS[table-format=1.2,round-mode=places]},
  table head={\toprule
    Person/Duck & Gender &
      {\makecell{Height\\ (\si{\metre})}}\\
    \midrule},
  head to column names=false,
  late after last line=\\\bottomrule,
  ]{test.csv}{Name=\myn, 2=\mys, 3=\myh,
    4=\myg}%
  {\myn\ \mys & \myg & \myh}
```

It may be that your text file has no header line giving the field names at all. In this case, we can use `head=false` or its abbreviation `no head` (this option cannot be used with `\csvautotabular` or similar automated commands).

Let us see an example processing a second text file which has no header line and semicolons instead of commas as separators:

testnohead.csv

```
van Duck, Paulinho;.4;M
de la Quack, Paulette;.35;F
Pescatore, Enrichetta;1.80;
Gregory, Henry;;M
```

[5] To tell the truth, it is a bit more complicated than this, but for our purpose, we do not need to be fussy.

Table 3: A table created with `\csvreader` applied to our semicolon-separated test file with no header line; column names are defined using the `table head` option.

N.	Person/Duck	Gender	Height (m)
1	van Duck, Paulinho	M	0.40
2	de la Quack, Paulette	F	0.35
3	Pescatore, Enrichetta		1.80
4	Gregory, Henry	M	

Please note that, since the separators are semicolons, we can have commas in the entry. Entries with commas would also be possible if the separators were commas, but in that case the entries must be surrounded by curly braces, for example:

```
{van Duck, Paulinho},.4,M
```

Generally, this can be done by the tool that produced the text file, or by any spreadsheet program.

The option `separator=`⟨*sign*⟩ indicates the separator character in the file, the possible values are: `comma`, `semicolon`, `pipe`, and `tab`.

The following macro gives the result shown in Table 3.

```
\csvreader[
  tabular={
    clcS[table-format=1.2,round-mode=places]
    },
  table head={\toprule
    N. & Person/Duck & Gender &
      {\makecell{Height\\ (\si{\metre})}}\\
    \midrule},
  nohead,
  separator=semicolon,
  late after last line=\\\bottomrule,
  ]{testnohead.csv}{1=\myn, 2=\myh, 3=\myg}%
  {\thecsvrow & \myn & \myg & \myh}
```

You can see that I have also used the convenient macro `\thecsvrow` to write the row numbers.

The above is plenty for basic usage, but the `csvsimple` package has many other features.

With `\csvset{`⟨*option list*⟩`}` you can set some options valid for all your `\csvreader` commands.

For example, if all your text files lack a header line and are separated by pipes, you can set this once for all as in the following, avoiding repeating them every time:

```
\csvset{
  nohead,
  separator=pipe
  }
```

`\csvstyle{`⟨*style name*⟩`}{`⟨*option list*⟩`}` allows you to create any customized style you need.

For example, you could create `mystyle`:

```
\csvstyle{mystyle}{
  tabular={
    clcS[table-format=1.2,round-mode=places]
    },
  table head={\toprule
    N. & Person/Duck & Gender &
      {\makecell{Height\\ (\si{\metre})}}\\
    \midrule},
  nohead,
  separator=semicolon,
  late after last line=\\\bottomrule}
```

and then conveniently use it in your `\csvreader` commands:

```
\csvreader[
  mystyle
  ]{testnohead.csv}{1=\myn, 2=\myh, 3=\myg}%
  {\thecsvrow & \myn & \myg & \myh}
```

There are also possibilities of filtering and sorting; further, you can use `\csvreader` not only for tables but also for any repetitive text, when the only things that change are the data in the text file. I will not discuss these features here, but I recommend reading the package documentation [1] for all the information.

4 Conclusions

I hope you enjoyed my explanation, and if you have any problem in processing a CSV file, remember:

Ask van Duck for a quack solution!

References

[1] Thomas F. Sturm. *The `csvsimple` package. Manual for Version 1.20 (2016/07/01).* `https://ctan.org/pkg/csvsimple`.

◇ Herr Professor Paulinho van Duck
Quack University Campus
Sempione Park Pond
Milano, Italy
`paulinho dot vanduck (at) gmail dot com`