Reproducible Research with Emacs Org-mode

[Demo]*

Karl Voit
Institute for Software Technology
Graz University of Technology
Austria
Karl.Voit@IST.TUGraz.at

Thomas S. Dye
Thomas S. Dye & Colleagues
735 Bishop St, Suite 315
Honolulu, Hawaii
tsd@tsdye.com

ABSTRACT

One important aspect of open science is the ability to reproduce results using the published data set. For this purpose it is crucial to use similar methods and tools as the original author producing the same result set. Reproducible research is a movement that tries to bridge this gap: within one single set of data one can not only find the raw data but also the methods and tools to process the data. The ultimate discipline is to complete this cycle from the raw data up to the presentation in the derived paper. Using a simple example this paper demonstrates how to combine raw data, scripts of various languages, and the describing text of a paper in one single file.

Categories and Subject Descriptors

I.7.1 [**DOCUMENT AND TEXT PROCESSING**]: Document and Text Editing—*Emacs*; I.7.1 [**DOCUMENT AND TEXT PROCESSING**]: Document Preparation; H.4.1 [**INFORMATION SYSTEMS APPLICATIONS**]: Office Automation—*Word processing*; I.7.4 [**DOCUMENT AND TEXT PROCESSING**]: Electronic Publishing; D.2.3 [**SOFTWARE ENGINEERING**]: Coding Tools and Techniques; D.4.9 [**OPERATING SYSTEMS**]: Systems Programs and Utilities; E.2 [**DATA STORAGE REPRESENTATIONS**]: Linked representations

Keywords

Open Science, Reproducible Research, Org-mode, Emacs, Tools

1. INTRODUCTION AND RELATED WORK

The editor Emacs[1] ([3]) provides a modular system that can be extended in various ways. One example of an extension is the Org-mode[2] which resembles a large set of features: outlining, task and project management, personal wiki system, export to a broad variety of formats, integration of external tools, and much more.

Using the Babel subsystem[3] to integrate external data sources – such as source code and execution results from these source code snippets – allows for creating meta-documents. This method can be applied for Literate Programming[4] or Reproducible Research[2, 5, 6].

Reproducible Research can be seen as the ultimate aim of Open Science[4] and Open Data[5]. People do not only get access to the data related to a research paper but also to the complete process of deriving the paper from the raw data up to the PDF file of the white paper.

This paper is a short demonstration of the general work flow from the very easy to write Org-mode text-file – which contains descriptive text and processing scripts – to the final PDF document in exact ACM template format[6].

Using a real world example, this paper develops a data processing path starting from raw data within comma-separated values (CSV) files up to visualizing plots. Different computer languages are part of this process: shell script tools, Python[7], and R[8].

Data is generated or processed using one computer language whose results are transferred to the next one which might be a different programming language from the previous. With this technique, it is possible to process the data with the most suitable command language appropriate for this single processing step. This is Literate Programming using multiple programming languages in linear sequence.

2. DOCUMENT GENERATION WORKFLOW

In order to use Org-mode to generate a PDF file (similar to this paper) the user has to follow two steps:

*The full source code of this paper is available on github `https://github.com/novoid/orgmode-iKNOW2012`

[1] `http://www.gnu.org/software/emacs/`

[2] `http://orgmode.org`

[3] `http://orgmode.org/worg/org-contrib/babel/intro.html`

[4] `http://en.wikipedia.org/wiki/Open_science`

[5] `http://en.wikipedia.org/wiki/Open_data`

[6] `http://www.acm.org/sigs/publications/proceedings-templates`

[7] Programming language, `http://www.python.org/`

[8] Statistical software, `http://www.r-project.org/`

1. Write the paper in Emacs Org-mode
   - much simpler syntax and complexity compared to LaTeX
   - easy outlining with hiding of sections
   - easy integration of external programming languages and commands and their output
   - everything in one single file
2. Export to LaTeX/PDF
   - a single command[9], directly invoked within Emacs or for example using GNU make

The TeX file is only *one* export file format of a larger number of possible output formats such as OpenDocument Text, DocBook, HTML, and so forth. Using this Org-mode template, the LaTeX export function automatically generates a LaTeX file and a PDF file which follows the ACM format.

This process simplifies the generation of white papers and enables others to completely re-produce any result mentioned in the paper. No other artifacts than this *single Org-mode file* (and external CSV files if used) are necessary to generate everything from scratch. And this resembles one of the most important advantages of the method presented here: a consistent, complete, and easy to re-generate form of the whole research contribution from raw data files up to the PDF paper result.

3. SHORT SYNTAX COMPARISON

The LaTeX markup language is complex in terms of "characters to type that are not visible in the result". The syntax of Org-mode is much more simpler, comparable to the reduced syntax languages of Wiki[10] systems. Therefore the Org-mode syntax is a lightweight markup language[11].

The following syntax example in both syntax languages demonstrates some aspects that make the difference syntax complexity obvious:

```
\begin{itemize}
\item First item inlcuding a \texttt{filename.txt}.
\item Second line with an
  \href{http://orgmodeorg}{URL of Org-mode}
\item An URL alone is even simpler:
  \url{http://tagtore.org}
  \begin{enumerate}
  \item indented enumerated list.
  \item This one contains \emph{emphasized} and
    \textbf{bold} words.
  \end{enumerate}
\end{itemize}
```

[9]Usually this is `org-export-as-pdf` mapped to `C-c C-e p`. However this paper is generated using a GNU make file in order to automate the whole process without having to modify the user configuration files. The command to generate this paper is `make all`.

[10]`http://en.wikipedia.org/wiki/Wiki`

[11]`http://en.wikipedia.org/wiki/Lightweight_markup_language`

```
The next part consists of a source
code\footnote{Using the \texttt{listings}
  package.} example:

\begin{lstlisting}[language=Python]
def hello_world():
    print "Hello World"

hello_world()
\end{lstlisting}

\begin{table}[ht]
  \centering
  \begin{tabular}{lr}
    \textbf{Input} & \textbf{Speed [s]} \\ \hline
    high dataset   & 4.1                \\
    low dataset    & 1.7
  \end{tabular}
  \caption{Performance results of the code.}
  \label{tab:results}
\end{table}
```

The LaTeX-content from above with the same markup in Org-mode syntax looks like following lines:

```
- First item inlcuding a ~filename.txt~.
- Second line with
  an [[http://orgmodeorg][URL of Org-mode]]
- An URL alone is even simpler: http://tagtore.org
  1. indented enumerated list.
  2. This one contains /emphasized/ and
     *bold* words.

The next part consists of a source
code[fn:lst] example:

#+BEGIN_SRC python :exports code
def hello_world():
    print "Hello World"

hello_world()
#+END_SRC

#+CAPTION: Performance results of the code.
#+LABEL:   tab:results
| *Input*        | *Speed [s]* |
|----------------+-------------|
| high dataset   | 4.1         |
| low dataset    | 1.7         |

[fn:lst] Using the ~listings~ package.
```

When Org-mode exports this code to PDF using LaTeX the results looks like this:

- First item inlcuding a `filename.txt`.
- Second line with an URL of Org-mode
- An URL alone is even simpler: `http://tagtore.org`

1. indented enumerated list.
2. This one contains *emphasized* and **bold** words.

The next part consists of a source code[12] example:

```
def hello_world():
    print "Hello World"

hello_world()
```

Table 1: Performance results of the code.

Input	Speed [s]
high dataset	4.1
low dataset	1.7

The simpler Org-mode syntax enables researchers to concentrate on the content without being distracted by expressing the form. If the user wants to use LaTeX commands that do not have an equivalent in Org-mode syntax, she can type in those LaTeX commands in the Org-mode file as well: the Org-mode exporter for LaTeX will pass those commands through.

4. EXAMPLE EVALUATION

So far the basic idea was described in previous sections. In this section, real-world data is used to produce a short research evaluation example.

In [7] the authors describe a formal experiment conducted with 18 test persons in the field of information retrieval. The original data set is available online[13].

We are using this public available data set in the following sections to demonstrate the analysis and visualization of the research question: to what extent differs the test persons average re-find task times within the usual folder hierarchy from average re-find task times using the newly proposed research software called *tagstore* [14] (see also [8]).

4.1 Reading in Data from Files

The analysis starts with reading in raw data from external files. In this first external CSV file, the re-finding performance of all test persons is stored. Since we are only interested in the task times, we want to filter out all other values that are irrelevant for this analysis. We are using GNU shell commands and tools to accomplish this task.

The following shell commands

1. reads in data from a CSV file,
2. removes all values before the character ";" (thus removing all values related to number of mouse clicks),
3. removes all incomplete lines (containing the string "TC"),
4. and removes the header line as well (using the `tail` command).

```
sed 's/.*;//' refinding_tagstore.csv | \
   grep -v "TC" | \
   tail -n +2
```

5.7	3.8	4.3	2.4	4.3	3.2
6.0	2.9	4.3	4.6	4.4	3.3
5.4	3.2	6.1	6.5	5.7	4.4
4.3	15.7	6.2	4.9	3.1	3.0
9.7	3.7	3.0	3.9	2.7	8.6
21.8	2.6	11.4	3.0	17.1	5.7
6.6	5.6	5.0	4.1	4.5	2.0
7.0	2.6	10.0	3.7	9.5	3.0
4.8	2.5	4.3	2.3	1.8	3.9
2.4	2.7	7.1	6.2	3.8	5.1
3.7	3.9	7.4	2.0	3.1	7.2
5.5	5.5	8.3	11.4	3.2	7.6
28.3	4.0	3.9	2.0	2.4	4.3
5.0	5.6	6.0	14.2	2.0	6.6
6.7	5.6	7.2	12.3	5.1	6.6

The table above resembles the direct output of the command provided above. Using Org-mode the user can decide, if the commands, or the output, or both gets printed in the exported file.

4.2 Generating Mean Values

Now that we have the raw task times of the test persons for the *tagstore* condition, we do want to derive the arithmetic mean values for further processing.

In the next step, the mean values will be calculated using the programming language *Python*. We take the result data from the last command from the previous section and use it as input data for the following source code snippet:

```
import numpy
return [round(numpy.average(row),2) for row in mytable]
```

This time, the output list consisting of 15 mean values is being suppressed for layout purposes.

4.3 Sorting Values

To demonstrate the combination of different command languages, we are now using GNU *tr* and *sort* to transform the list of mean values in a column format of mean values and sort it reverse numerical:

```
echo ${myvalues} | tr ' ' '\n' | sort -nr
```

We get the sorted list of mean values of the task times for all test persons in the *tagstore* condition.

[12] Using the `listings` package.
[13] `https://github.com/novoid/2011-01-tagstore-formal-experiment`
[14] `http://tagstore.org`

4.4 Process Folder Values

In this section we repeat the previous steps with the task data for the second test condition, the folder condition.

```
sed 's/.*;//' refinding_folders.csv | \
   grep -v "TC" | \
   tail -n +2
```

6.7	5.4	2.4	3.9	3.6	3.8
5.4	3.1	3.4	3.5	3.3	3.6
6.5	6.6	4.0	4.4	4.0	5.1
3.0	3.3	3.7	7.1	2.8	4.3
6.6	3.6	10.3	4.6	5.4	3.7
2.7	3.2	9.4	18.0	4.7	3.8
7.0	3.7	8.1	4.9	5.2	5.2
34.1	2.8	8.9	8.9	3.1	8.3
4.0	2.9	3.6	5.7	5.0	5.5
4.8	1.4	3.5	3.5	3.3	1.9
42.9	1.9	12.3	5.8	7.6	3.4
7.0	5.2	5.0	3.8	5.1	4.2
19.3	1.6	11.9	7.0	3.9	4.0
6.6	6.6	4.6	7.5	3.8	5.2
6.0	3.2	5.1	4.4	5.9	4.0
4.6	1.6	3.4	4.1	4.4	3.8
7.1	4.5	7.0	7.6	5.5	7.5

```
import numpy
return [round(numpy.average(row),2) for row in mytable]
```

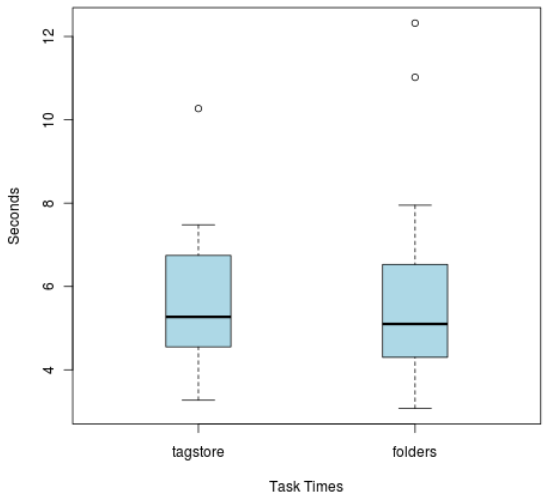
```
echo ${myvalues} | tr ' ' '\n' | sort -nr
```

We do now have the second list of average task times which relates to the folder condition of the formal experiment.

4.5 Plotting Data

The open source statistical software R offers many different kind of commands to process data. To visualize the two sets of data using a boxplot graph, we are using the following R script:

```
png('my_boxplot_data.png')
## following two commands compensate a bug
## in the Babel method which should be fixed soon:
mFdata=c(4.3, 3.72, 5.1, 4.03, 5.7, 6.97, 5.68,
         11.02, 4.45, 3.07, 12.32, 5.05, 7.95,
         5.72, 4.77, 3.65, 6.53)
mTSdata=c(3.95, 4.25, 5.22, 6.2, 5.27, 10.27,
          4.63, 5.97, 3.27, 4.55, 4.55, 6.92,
          7.48, 6.57, 7.25)
boxplot( list(mTSdata, mFdata),
    names=c("tagstore", "folders"),
    xlab="Task Times", ylab="Seconds",
    pars = list(boxwex = 0.3, staplewex = 0.5,
    boxfill="lightblue"))
```



The resulting graph visualizes the overall result of the two test conditions: the average task times in the *tagstore* condition and the folders condition.

Every single step from the raw data from the CSV file to the graph can be completely checked and re-created by anybody who wants to do so. This method of Reproducible Research guarantees the highest level of transparency and confirmability.

5. OVERVIEW

Table 2: Overview of the input values, execution languages, and output values.

Input	Using	Output
`refinding_tagstore.csv`	shell	TS task time values
TS task time values	Python	TS avr. time values
TS avr. time values	shell	TS sorted numbers
`refinding_folder.csv`	shell	F task time values
F task time values	Python	F avr. time values
F avr. time values	shell	F sorted numbers
avr. time values TS + F	R	boxplot of times

To sum up the different data paths, we conclude in Table 2 the different steps. We have read in raw data from external CSV files and filtered them using shell tools.

Then we processed the data using a Python program in order to compute the average mean values of the times. Using the tool "sort" we got the sorted list of mean values which were visualized using R into boxplot graphs.

To summarize, the method of writing research papers within Org-mode can simplify the writing process itself, lets the author choose the most suitable tool for the next data processing step, create additional value by having an easy to invoke data processing method, and it illustrates the author's work in the most transparent way possible.

6. LIMITATIONS

For the purpose of this paper, the Org-mode LaTeX export preferences were adopted in order to meet the specifications of the ACM paper template. To be future-proof, the authors used the upcoming new LaTeX exporter that will replace the current one. This leads to following current limitations.

The normal Org-mode LaTeX PDF export command has to be replaced by the command for the command `org-e-latex-export-to-latex` for the enhanced LaTeX PDF export.

Usually, each Emacs user has Emacs configuration files that are highly personalized. For generating the PDF export file independent of the current user configuration, a stand-alone Emacs configuration file was created.

Having to adopt to the specifications to the ACM template a number of settings had to be altered. There were still some issues that could not be solved yet such as remaining TeX lines for `author-list` and such. Those unnecessary lines have to be removed from the TeX file after the LaTeX export and before the compile process to generate the PDF file. All those steps are encapsulated within a GNU make file. The single command line sequence `make all` generates everything from the Org-mode file to the final PDF file.

7. SUMMARY AND OUTLOOK

This paper demonstrates a new way of generating research papers directly from the outlining tool Org-mode which is an enhancement of the Emacs editor. The outlining file contains not only the text from the paper but also all processing steps and scripts. A method like this allows reproducible research from the raw data files up to the results generated without leaving any gap in between.

Since this method is relatively new and is using many customizations for ACM template format, a few workarounds described in the previous section were necessary. With future versions of Org-mode, those workarounds will be minimized or removed totally.

8. REFERENCES

[1] *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, Austin, Texas, USA, May 2012. ACM.

[2] M. Delescluse, R. Franconville, S. Joucla, T. Lieury, and C. Pouzat. Making neurophysiological data analysis reproducible. why and how? *Journal of Physiology Paris*, (0), Aug. 2011.

[3] C. Dominik. *The Org-Mode 7 Reference Manual: Organize Your Life with GNU Emacs*. Network Theory, UK, 2010. with contributions by David O'Toole, Bastien Guerry, Philip Rooke, Dan Davison, Eric Schulte, and Thomas Dye.

[4] D. E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[5] E. Schulte and D. Davison. Active documents with org-mode. *Computing in Science Engineering*, 13(3):66 –73, June 2011.

[6] E. Schulte, D. Davison, T. Dye, and C. Dominik. A multi-language computing environment for literate programming and reproducible research. *Journal of Statistical Software*, 46(3):1–24, 1 2012.

[7] K. Voit, K. Andrews, and W. Slany. TagTree: Storing and re-finding files using tags. In *Proc. 7th Conference of the Austrian Computer Society Workgroup: Human-Computer Interaction (Usab 2011)*, volume 7058 of *LNCS*, pages 471–481. Springer, Nov. 2011.

[8] K. Voit, K. Andrews, and W. Slany. Tagging might not be slower than filing in folders. In *International Conference of Computer-Human Interaction 2012 (CHI2012)* [1].