Invertible Reproducible Documents

Eric Lim jlim062@aucklanduni.ac.nz[1], Paul Murrell paul@stat.auckland.ac.nz[1], and Finlay Thompson finlay@dragonfly.co.nz[2].

[1]Department of Statistics, University of Auckland
[2]Dragonfly Science

24 July 2014

Abstract

abstract text here

Motivation

This article addresses the following standard scenario: a *consultant* needs to prepare a report for a *client* where, in addition to normal text, the report contains tables and graphs based on the results of data analysis.

Following best practice, the consultant produces the report as a *reproducible document*, using tools such as 'Sweave' (Leisch, 2002) and 'knitr' (Xie, 2014). This means that the consultant creates and edits a *source document* that combines normal text content with sections of computer code - *code chunks* - where the code chunks describe data analysis tasks. The figure A source document presents a minimal example of a reproducible document.

```
<html>
  <body>
    <p>Normal text</p>
<!--begin.rcode echo=FALSE
Sys.Date()
end.rcode-->
  </body>
</html>
```

A source document: This HTML-based reproducible document contains a combination of normal HTML content and a chunk of R code.

The source document is then processed to produce a *final document* for the client. Part of this processing involves running the code chunks and inserting the results, e.g., tables and graphs, in the final document. The figure A final document presents the result from processing the document in figure A source document.

```
<html>
<body>
  <p>Normal text</p>
  <div class="chunk" id="unnamed-chunk-15">
    <div class="rcode">
      <div class="output">
        <pre class="knitr r">
[1] "2014-07-24"
</pre>
      </div>
    </div>
  </div>
</body>
</html>
```

Normal text

```
[1] "2014-07-24"
```

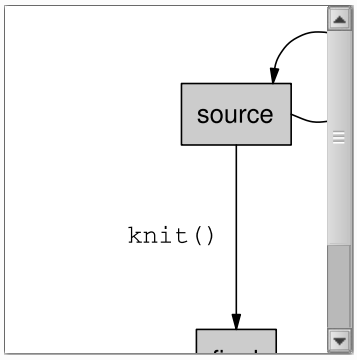
A final document: This HTML document is the result of processing the HTML-based reproducible document in figure A source document. On the left is the raw HTML and on the right is the final document as it appears in a web browser; the code chunk in the source document has produced a date in the final document.

This workflow benefits the consultant because regenerating the final document is very simple, fast, and reliable. There is only one source document where all changes are made and all changes to the source document are automatically incorporated into the final document during processing.

However, a problem arises when the client is asked to provide comments on the report. Typically, the source document and the final document have different formats. For example, the source document is a TeX file and the final document is a PDF file, or the source document is a markdown file and the final document is an HTML file. Depending on the format of the final document and the software available to the client, it may not be possible for the client to make direct modifications, in which case comments may only be hand-written on a hard-copy of the report. Where the final document can be edited, changes to the final document format will not usually map in a simple way to changes to the source document. In either case, the task of merging comments from the final document back into the source document is very laborious and error-prone.

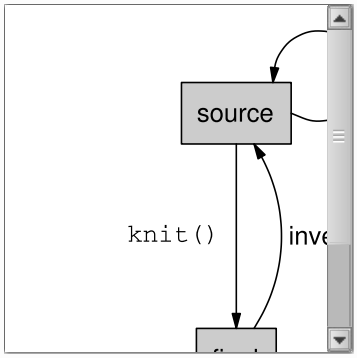
This article describes an experiment to set up a reproducible document workflow that allows changes made by a client to the final document to be merged accurately and efficiently back into the source document.

As a bird's-eye view, the Figure 'knitr' workflow shows the basic outline of a standard reproducible document workflow. The consultant works on a source document then processes it with 'knitr' to produce a final document for the client.



'knitr' workflow: The standard uni-directional workflow from source document to final document using a reproducible document tool like 'knitr'.

The Figure Inverted workflow (high level) shows the basic outline of the experimental workflow. The consultant works on a source document then processes it with 'knitr' to produce a final document for the client, *plus* the client is able to make changes to the final document, *plus* it is possible to process the final document to get back to the source document format, with the client's changes incorporated.



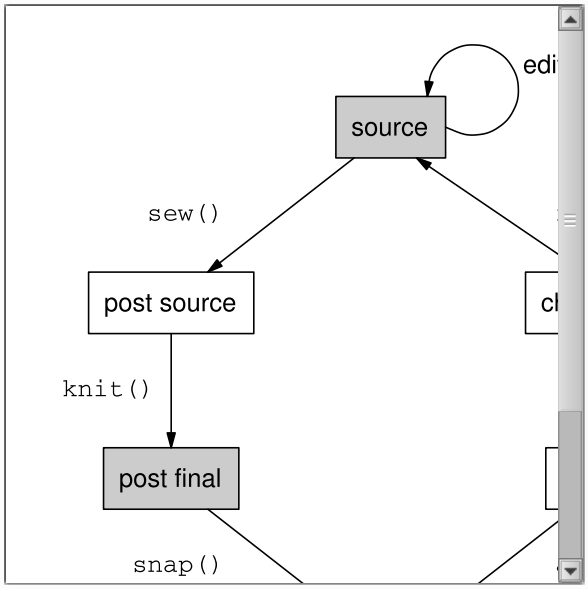
Invertible workflow (high level): An invertible workflow from source document to final document and back again.

The advantage of the invertible workflow is that changes by the client are embedded directly within a new source document. This means that the consultant is saved the work of merging client changes. It also means that client changes can be viewed as just a new version of the source document, which has positive implications for version control. The Section Discussion explores the advantages and disadvantages of the experimental workflow in much greater detail.

Solution overview

This section provides a brief outline of the important steps involved in the proposed workflow. The Figure Invertible workflow (low level) shows a diagram of the individual steps and the purpose of each step is described below. The Section Solution details provides more detailed descriptions of each step.

`sew()`
: An extra pre-processing step is introduced before processing the source document with 'knitr'. The purpose of this step is to take a copy of all code chunks in the source document. These copies are retained to assist with the conversion from final document back to source document.



Invertible workflow (low level): An invertible workflow from source document to final document and back again.

Solution details

Discussion

Beyond R

The discussion of this work has been centred around reproducible tools in the R environment, but reproducible documents and shared documents exist in many other forms as well. Compare with other reproducible document tools, e.g., odfWeave. One different is that ODF might make it easier for the end user to make edits, but it is NOT the format that the consultant/developers want to be working in! Ditto Word; so the benefit of having a native GUI for the document format (so source and final document are the same format) is offset by having to force the consultant to work in a horrible GUI. Developer would prefer to work in a world of text-based document formats, where developer tools can be brought to bear and documents can be worked with programmatically. There are can also be image format issues. For example, odfWeave dictates that images must be in PNG format(!? - actually that may just be the DEFAULT setting). The choice of document format generally has an impact on things like available image formats, support for mathematical formulas, ... Compare with the TeX/PDF tool that lets you click on the PDF and see where in the TeX that PDF came from. Useful for developers more than for client-consultant communication? This is NOT direct PDF editing that is reflected in source TeX. Compare with IPython Notebooks, wikis, and cloud documents. Is there a connection with Continuous Integration here too? Note that the IPython Notebooks site says "While in simple cases you can “roundtrip” a notebook to Python, edit the Python file, and then import it back without loss of main content, this is in general not guaranteed to work.". One difference with wikis and cloud documents is the inclusion of runnable code chunks(!) Though see the wikirobot package (though inactive for 4 years) and the R Extension for MediaWiki (including installation instructions, documentation and examples; active in 2014!). Another difference with Wikis is that they do less sophisticated version control (than git) ?

Version control

The importance of 'diff'

We have worked quite hard to make sure that the ONLY differences between the original source file and an inverted file are differences that have been deliberately introduced by editing or annotating the final document. In particular, we have worked hard to avoid any spurious changes in white space, including changes in indenting and changes in where line breaks occur.

This emphasis is justified by the fact that it is then possible to use simple, standard, text processing tools, e.g., 'diff', to easily identify changes. This has major flow-on benefits for version control because tools like 'git' and 'svn' can then reliably track the changes. More complex situations where developer changes to the source document occur concurrently with client changes, which require a merge of different versions, can also be handled with existing version control tools.

XML versus text processing

XML is much better and more reliable and more powerful, but, at least using the tools that we used, it tends to generate spurious changes in white space and you run into encoding issues (and issues with special entities).

Reusable documents versus reproducible documents

A general lesson is that tools for processing a document from one format to another should ideally do so in as clean and predictable way as possible. If this approach is taken then the processed document can be used for more than just viewing.

This is the approach taken in 'gridSVG' by the way.

Acknowledgements

This article was processed from a source document using 'knitr' and 'knitcitations' (Boettiger, 2014).

References

[1] C. Boettiger. *knitcitations: Citations for knitr markdown files*. R package version 1.0.1.99. 2014. URL: https://github.com/cboettig/knitcitations.

[2] F. Leisch. “Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis”. In: *Compstat 2002 — Proceedings in Computational Statistics*. Ed. by W. H<U+00E4>rdle and B. R<U+00F6>nz. ISBN 3-7908-1517-9. Physica Verlag, Heidelberg, 2002, pp. 575–580. URL: http://www.stat.uni-muenchen.de/~leisch/Sweave.

[3] Y. Xie. *knitr: A general-purpose package for dynamic report generation in R*. R package version 1.6. 2014. URL: http://yihui.name/knitr/.