LaTeX, Sweave, and SVN: An introduction to writing, maintaining, and sharing structured documents

Phil Chalmers

York University

November 21, 2011

Abstract

Abstract

In principle there are two types of documentation strategies: 'What You See Is What You Get' (WYSIWYG), and 'What You See Is What You Mean' (WYSIWYM). The former strategy is adopted by commercial programs such as Microsoft Word, OpenOffice, and Word Perfect, while the latter is found in markup languages such as HTML, XML, and (La)TeX. While WYSIWYG styles are certainly more popular, a WYSIWYM driven style can offer a powerful, flexible, cross-platform, and free alternative. In this lecture we will explore several aspects of the LaTeX documentation system that make it appealing for scholarly writers, demonstrate how LaTeX can interface with R directly by using Sweave, and show how collaborating with multiple authors on a single document can be facilitated by using SubVersioN (SVN) techniques.

redefine THE POSSIBLE.

Programs Used

I'll be using a Windows OS to demonstrate these techniques using 3 programs, but many other programs could have been used instead. Primarily, I will be using

Rstudio	A powerful IDE that supports R and LaTeX
TortoiseSVN	A GUI client for using SVN techniques
Dropbox	An open-source program that allows 2GB+ of server sided storage which allows for specific folder sharing, conflict resolutions, and time stamped syncing

All of these can be found quickly with a Google search and are free to use. The only program that isn't cross-platform is TortoiseSVN, so Mac OS and Linux users will have to find a different SVN client (or use the build in command line functions).

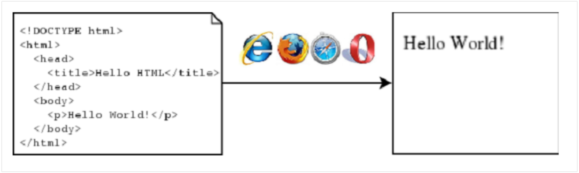
LaTeX

What is LaTeX?

LaTeX is a macro driven document preparation system created by Leslie Lamport to simplify the popular TeX typesetting system created by Donald Knuth. Essentially, LaTeX is a language used to declare how a document should be organized and structured. Like the wide-spread statistical programming language R, LaTeX is

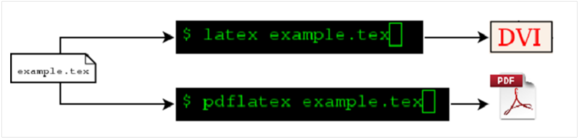
- free to use
- supports different mediums (manuscripts, books, presentations, . . .)
- open-source
- cross-platform, and
- extensible (CTAN)

LaTeX versus HTML



```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello HTML</title>
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>
```

Figure: HTML interpreted by browser for the final product



```
$ latex example.tex
$ pdflatex example.tex
```

Figure: LaTeX requires a compiler to create the final product

Why should I use LaTeX?

LaTeX is not for everyone, and in fact may not be the most efficient tool for every project. Like R, LaTeX has a learning curve that is initially hard to clime. However, there are several enticing aspect that are ideal for researchers who write structured documents, namely:

- output is professional, consistent, and formatted globally
- local formatting often carries over to future similar projects and tasks
- project can be broken into several files (intro.tex, appendix.tex, . . .)
- mathematical typesetting can be easily written and organized
- bibliography (via BibTeX) integration is simple and also cross-platform
- many different TeX editors available to suit the users preferences (bibliography management included)
- allows direct integration with R via Sweave, and
- can benefit from SVN techniques to track document changes

Why should I use LaTeX?

LaTeX is not for everyone, and in fact may not be the most efficient tool for every project. Like R, LaTeX has a learning curve that is initially hard to clime. However, there are several enticing aspect that are ideal for researchers who write structured documents, namely:

- output is professional, consistent, and formatted globally
- local formatting often carries over to future similar projects and tasks
- project can be broken into several files (intro.tex, appendix.tex, . . .)
- mathematical typesetting can be easily written and organized
- bibliography (via BibTeX) integration is simple and also cross-platform
- many different TeX editors available to suit the users preferences (bibliography management included)
- allows direct integration with R via Sweave, and
- can benefit from SVN techniques to track document changes

My very first LaTeX document!

LaTeX files largely work within predefined *environments*, which must be properly opened and closed. The main text file (.tex) also requires a document class, the most popular being the 'article'. Optional arguments to many environments are enclose in [ ], and here we globally change the font size default to 12pt.

Example

```
\documentclass[12pt]{article}
\begin{document}
Hello world!
\end{document}
```

When compiled with `pdflatex` from the console, or compiled with a dedicated IDE (we'll use Rstudio), creates . . .

Special characters

commands	start with a \, some commands require parameters between {. . . }, and optional arguments are declared within [. . .]
spaces	consecutive spaces are treated as only *one* space, and new paragraphs are made by inserting a full blank line (hit 'Enter' twice)
special	# $ % & _ { } have special meanings, but can be written by using a backslash. The ∼ character acts as a 'glue' operator so that two words always appear on the same line
comments	anything following a % is interpreted as a comment and does not show up in the final document

Preamble

Global formatting, document variables, user define commands, and packages are inserted before \begin{document}. Here we include a class file that I wrote for creating manuscripts in APA 6:

Example

```
\documentclass{apa6} %My APA6 package
\title{My Title}
\runninghead{My running head}
\author{Phil Chalmers \\ York University}
\authornote{This is a test note.}
\abstract{Here is an abstract \ldots}
\keywords{and some keywords}
```

Which gives . . .

Declaring the structure

- LaTeX offers several document ‘levels’, such as `\part{...}`, `\chapter{...}`, `\section{...}`, `\subsection{...}`, etc., but for articles we really only need from sections on.
- Tasks like generating indexes, table of contents, and a list of figures are made much easier since details are often automatically constructed without the user even knowing, and only require declaring special calling commands such as `\tableofcontents` and `\listoffigures`.
- Bibliographies section are automatically constructed and will only contain reference that actually occurred within the document (using commands such as `\cite{author}`, `\citep{author}`, etc.). These can be declared internally, or called from external (perhaps multiple) .bib files.

Building tables

Table are built in LaTeX using the `tabular` environment, where columns are separated by &, new lines are indicated by \\. Column justifications and line separators appear in the curly brackets immediately after the opened `tabular` environment.

```
\begin{tabular}{|r|l|}
  \hline
  7C0 & hexadecimal \\
  3700 & octal \\ \cline{2-2}
  11111000000 & binary \\
  \hline \hline
  1984 & decimal \\
  \hline
\end{tabular}
```

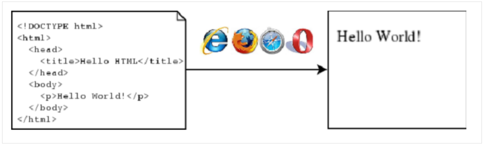
7C0	hexadecimal
3700	octal
11111000000	binary
1984	decimal

Inserting figures

Typically .png, .jpg, and .pdf files can be imported as figures when using a `pdflatex` compiler. Cropping, resizing, and rotating can be optionally declared so that no modification are made directly to the original image. Here I've stored the image 'htmlhelloworld.png' in a subdirectory just below where I kept my main .tex file called 'img'.

Example

```
\begin{figure}
  \includegraphics[width=2.5in]{img/htmlhelloworld}\\
  \caption{HTML interpreted by browser for the final product}
\end{figure}
```



```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello HTML</title>
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>
```

Figure: HTML interpreted by browser for the final product

Equations

Equations can also be created very easily, and have their own special grammar.

Example

```
A quadratic linear model is written as:
\begin{equation} \label{eqn:linearmodel}
    y_i = \alpha + \beta_1 X_i + \beta_2 X_i^2.
\end{equation}
In Equation~\ref{eqn:linearmodel} the $\beta_j$ terms \ldots
```

Which gives . . .

A quadratic linear model is written as:

$$y_i = \alpha + \beta_1 X_i + \beta_2 X_i^2. \tag{1}$$

In Equation 1 the β_j terms . . .

Learning more!

We have only touched on some of the basic tasks that LaTeX does, so if you're interested (it's free to try after all!) check out some of the following sites:

A great Wikibook `http://en.wikibooks.org/wiki/LaTeX/Introduction`

Not so short guide to LaTeX `http://tobi.oetiker.ch/lshort/lshort.pdf`

An HTML Intro `http://www.maths.tcd.ie/~dwilkins/LaTeXPrimer/`

As well, there are several books available from Amazon.ca that are very helpful, ranging from beginner to advanced. Getting started on Windows OS requires downloading and installing MiKTeX, Mac OS requires MacTeX, and Linux systems require the TeXLive bundle.

Sweave

Sweave

Using a method known as 'Sweaving' R code can be placed directly inside of a LaTeX document to produce consistent and reproducible results. Here are the basic steps:

1. Write a document in a plain text file, just as you would with LaTeX. This will look like a LaTeX document with "code chunks" of R mixed in. Name it doc.Rnw.
2. Run `Sweave('doc.Rnw')` in R to process the code and produce doc.tex.
3. Run pdflatex (or some other compilation method) to produce a doc.pdf output.

Thankfully . . .

Because of programs like Rstudio this has never been easier since these steps are done for us automagically!

Sweave

Using a method known as 'Sweaving' R code can be placed directly inside of a LaTeX document to produce consistent and reproducible results. Here are the basic steps:

1. Write a document in a plain text file, just as you would with LaTeX. This will look like a LaTeX document with "code chunks" of R mixed in. Name it doc.Rnw.
2. Run `Sweave('doc.Rnw')` in R to process the code and produce doc.tex.
3. Run pdflatex (or some other compilation method) to produce a doc.pdf output.

Thankfully . . .

Because of programs like Rstudio this has never been easier since these steps are done for us automagically!

Sweave Example

Here is a simple example of a Sweave 'code chunk': Generate 100 random normal values and print them all with only 2 decimal places.

Example

```
<<>>=
x <- rnorm(100)
round(x,2)
@
```

What's interesting is that these results can be reused, so 'x' can be called at a later time for perhaps another purpose. Let's try it ...

More Sweave Examples

Sweave can be controlled by passing options to tell it what to do. By default, Sweave prints the source code, evaluates and prints the results, and doesn't generate any figures. These can be overruled by adding the options:

eval=false	don't evaluate the R code, just display the source code
echo=false	don't show the source code, just print the console results
fig=true	print a figure (only one allowed per code chunk)
results=tex	treat the code produced by `xtable` as results to be interpreted by the TEX compiler

So . . .

Even More About Sweave!

There are two other interesting aspect of Sweave that are useful for creating consistent documents. The first is that single values that have been created can be displayed inside text and called by using `\Sexpr{Robject}`. For instance, say we want only the first value of x to show up in-text with only two decimal places. We can do this by writing '`The first value of `x` is \Sexpr{x[1]}`', which when compiled would read 'The first value of x is -0.11'.

Sweave and I

Sweave is excellent when we want to make our source code public, and is great for showing off and describing functions in R. In fact, and perhaps not surprisingly at this point, R's vignette files are written with Sweave. However, many people will have little use for this aspect of Sweave. For example, say we analyzed some data and want to present our results and have no intention of displaying our code. In this case I would use the following strategy:

1. Create an R workspace file with the object formatted to the number of decimals you want them to be displayed
2. Load that workspace into the beginning of your .Rnw file using `load('the/file/location.rdata')`
3. Use `\Sexpr{Robject}` and the `result=tex` option at you leisure now without worrying about formatting

SubVersioN

What is SVN?

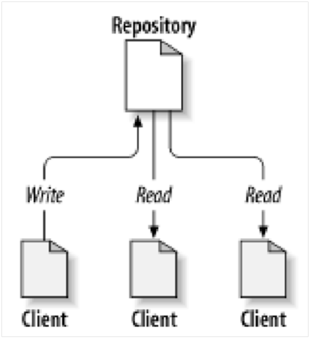


Figure: Repository shared by 3 authors.

Unlike the 'pass the ball' method of sharing files, where only one master file is passed around at a time, SVN is an attempt to create a centralized 'master collect' that can be read and contributed to simultaneously. Mostly, it's been a popular method used by software developers for maintaining source code files. The central idea of SVN is the 'repository', which is essentially the collection of files that are being constructed, and is available to all authors to read and write to.

SVN method of sharing documents

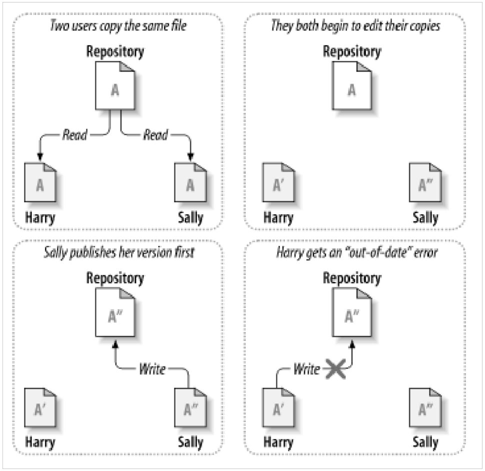

Two users copy the same file
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
Repository
A
A′
A″
Harry
Sally
Sally publishes her version first
Repository
A″
Write
A′
A″
Harry
Sally
Harry gets an "out-of-date" error
Repository
A″
Write
A′
A″
Harry
Sally

Figure: 'Out-of-date' error stopping any accidental overwriting.

SVN method of sharing documents (continued) . . .

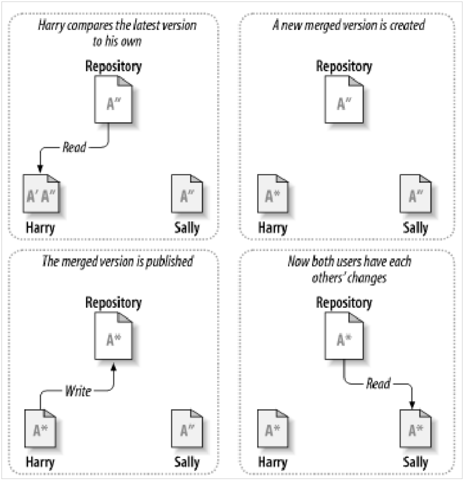
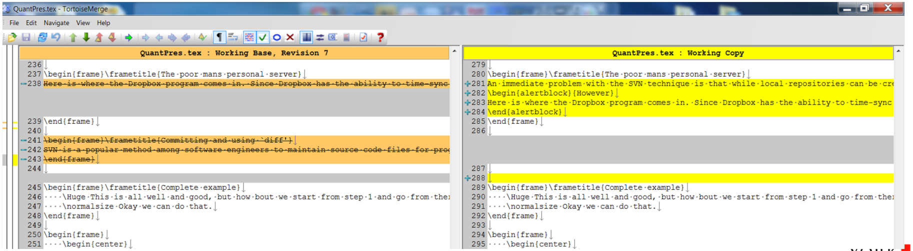

Harry compares the latest version to his own
Repository
A″
Read
A′ A″
Harry
A″
Sally
A new merged version is created
Repository
A″
A*
Harry
A″
Sally
The merged version is published
Repository
A*
Write
A*
Harry
A″
Sally
Now both users have each others' changes
Repository
A*
Read
A*
Harry
A*
Sally

Figure: . . . a new merged document is then created, and new results are committed and published.

Versioning, Committing, and using 'diff'

A great benefit of SVN is that all changes to files after they are 'committed' are tracked and can be reverted to or explored at any time. This means that even changes made at the beginning of a project can be rediscovered if necessary. Also note that if there are conflicts between documents a process called *diff* is initiated to help detect where these conflicts are.



QuantPres.tex : Working Base, Revision 7

```
 
 \begin{frame}\frametitle{The poor mans personal server}
-Here is where the Dropbox program comes in. Since Dropbox has the ability to time-sync
 \end{frame}
 
-\begin{frame}\frametitle{Committing and using `diff'}
-SVN is a popular method among software engineers to maintain source code files for proc
-\end{frame}
 
 \begin{frame}\frametitle{Complete example}
     \Huge This is all well and good, but how bout we start from step 1 and go from the
     \normalsize Okay we can do that.
 \end{frame}
 
 \begin{frame}
     \begin{center}
```

QuantPres.tex : Working Copy

```
 
 \begin{frame}\frametitle{The poor mans personal server}
+An immediate problem with the SVN technique is that while local repositories can be cre
+\begin{alertblock}{However}
+Here is where the Dropbox program comes in. Since Dropbox has the ability to time-sync
+\end{alertblock}
 \end{frame}
 
 
+
 \begin{frame}\frametitle{Complete example}
     \Huge This is all well and good, but how bout we start from step 1 and go from the
     \normalsize Okay we can do that.
 \end{frame}
 
 \begin{frame}
     \begin{center}
```

The poor man's personal server!

An immediate problem with the SVN technique is that while local repositories can be created and accessed (using a file:// path) on any system for personal use, we really need a method to allow other users access to the repository. Traditionally, this is done by setting up a server sided location that users can access at any time using an http://, svn://, or svn+ssh:// connection.

Fortunately

This is where the Dropbox program comes in to play. Since Dropbox has the ability to time-sync any files that are newer than the ones currently stored on the server, and can share folders and files using 'Dropbox Invites', we can use it as our own personal server for syncing and sharing our local repository. This creates a controlled and secure system that is only shared between selected authors.

The poor man's personal server!

An immediate problem with the SVN technique is that while local repositories can be created and accessed (using a file:// path) on any system for personal use, we really need a method to allow other users access to the repository. Traditionally, this is done by setting up a server sided location that users can access at any time using an http://, svn://, or svn+ssh:// connection.

Fortunately

This is where the Dropbox program comes in to play. Since Dropbox has the ability to time-sync any files that are newer than the ones currently stored on the server, and can share folders and files using 'Dropbox Invites', we can use it as our own personal server for syncing and sharing our local repository. This creates a controlled and secure system that is only shared between selected authors.

Setting up a repository

Setting up a repository with Dropbox is surprisingly simple.

1. Create an empty folder in your Dropbox location, right-click → TortoiseSVN → ‘Create repository here’, copy the location and click ‘OK’
2. Add the files and folders that you want to be SVN’ed by right-clicking on the highest level of the folder and selecting TortoiseSVN → ‘Import’ using the copied file:// path as the target location
3. Right-click anywhere to ‘SVN checkout’ the repository given the copied file:// path (this is your version for editing and committing changes)
4. (optional) Share the folder with right-click → Dropbox → ‘Share this folder’

Complete example

That's all well and good, but how 'bout we start from step 1 and go from there? Okay we can do that.

THANK YOU.