Glisterings

Peter Wilson

> . . . Cloath'd all in glistering coats, which made a shew . . .
>
> *Poems and Fancies*, Margaret Cavendish

The aim of this column is to provide odd hints or small pieces of code that might help in solving a problem or two while hopefully not making things worse through any errors of mine.

Corrections, suggestions, and contributions will always be welcome.

> Sir, I have found you an argument, but I am not obliged to find you an understanding.
>
> Samuel Johnson

1 Verbatim arguments

I have been reminded recently that one problem with verbatim material is that it cannot be used in an argument to a regular command (or environment). For example to typeset something in a framed `minipage` the obvious way is to use the `minipage` as the argument to the `\fbox` macro:

```
\fbox{\begin{minipage}{0.97\columnwidth}
      Contents of framed minipage
      \end{minipage}}
```

This works well until the contents includes some verbatim material and then you get nasty messages, even though it appears to be wrapped inside the `minipage`.

However, we can put material into a box, declared by `\newsavebox`, and output the typeset contents later on via `\usebox`. This is how the framed text below was processed.

This is the definition of the `framedminipage` environment which lets you put verbatim text into a frame. All this is set within a `framedminipage` to show that it does work.

```
\newsavebox{\minibox}
\newenvironment{framedminipage}[2][c]{%
  \begin{lrbox}{\minibox}
     \begin{minipage}[#1]{#2}}%
     {\end{minipage}\end{lrbox}
     \fbox{\usebox{\minibox}}}
```

I used `0.97\columnwidth` as the width of the environment like this:

```
\begin{framedminipage}%
      {0.97\columnwidth}
...
```

An `lrbox` is an environment form of a `\savebox` (or `\sbox`) and we can use it to solve the framed minipage problem. The code displayed above, after getting a new save box (`\minibox`) defines a `framedminipage` environment which is used just like a regular `minipage`, including the optional positioning argument. It starts by opening an `lrbox` environment, then a `minipage` environment. At the end it closes the `minipage` and `lrbox` environments and then typesets an `\fbox` whose argument is the saved box *the contents of which have already been typeset, verbatims and all.*

In *The TEXbook*, page 363, there is code for a `\footnote` macro that can take verbatim material in its argument. Knuth says that it is subtle and requires trickery, and I don't understand it, but here is the essence, in the form of a one argument macro I've called `\verbtext`. I'm not sure, though, about the location of the `\color@...` macros as there was nothing comparable in Knuth's original code

```
\makeatletter
\long\def\verbtext{\vtintro\futurelet\next\vte@t}
\def\vte@t{\ifcat\bgroup\noexpand\next
                \let\next\vt@@t
          \else \let\next\vt@t\fi \next}
\def\vt@@t{\bgroup\aftergroup\vtend\let\next}
\def\vt@t#1{%
  \color@begingroup
  #1\vtmid
  \color@endgroup}
\let\vtintro\relax
\let\vtmid\relax
\let\vtend\relax
\makeatother
```

The macros `\vtintro` and `\vtend` are called before and after the argument is read and you can try and define them to do something you think is useful. Defining `\vtmid` may, on occasion, be helpful.

So, here is an example of the `\verbtext` command, which can take verbatim text as part of its argument.

```
\verbtext{'The argument to \verb?\verbtext?
          can include \verb?\verb? text.'}
```

'The argument to `\verbtext` can include `\verb` text.'

The following code is a simple example of using `\vtintro` and `\vtend` to specify a small caps font.

```
\makeatletter
\newcommand*{\fred}[1][\@empty]{Frederick%
             \ifx\@empty #1\else\ #1\fi}
\makeatother
\def\vtintro{\begingroup\scshape}
\def\vtend{\endgroup}
\verbtext{The macro \verb?\fred[III]?
          produces \fred[III], while
          \verb?\fred? results in \fred.}
```

The macro `\fred[III]` produces Frederick III, while `\fred` results in Frederick.

Actually this could have been done as easily as:

```
{\scshape\verbtext{...}}
```

without bothering to redefine `\vtintro` and `\vtend`, but perhaps you may come across occasions when they can help in solving a particular problem.

> Wickedness is always easier than virtue; for it takes a short cut to everything.
>
> Samuel Johnson

2 Cut off in its prime

Changing the subject, there was a question posed on `comp.text.tex` asking if there was any way of cutting a long text short, such as after two or three lines.

Donald Arseneau's `truncate` package [1] is available for truncating text to a specified width. By default ... (`\ldots`) is typeset at the end of the truncated text to indicate that something is missing. For instance

```
\truncate{0.9\columnwidth}{The
 \texttt{truncate} package provides a macro
  for cutting off text so that it does not
  exceed a given length.}
```

will result in:

The `truncate` package provides a macro for ...

However, in response to the query Donald came up with a vertical equivalent to `\truncate` which he called `\vtruncate` [2], as follows:

```
\newsavebox\descbox
\newsavebox\partialbox
\newcommand{\vtruncate}[2]{%
  \setbox\descbox\vbox{{#2\par}}%
  \setbox\partialbox\vsplit\descbox to #1\relax
  \vtop{\unvbox\partialbox}%
% or use
%   \par\unvbox\partialbox
}
```

The first argument is the vertical space and the second is the text.

Will Robertson also responded, but with an environment, `cutlines`, that would truncate its contents if it exceeded a certain height [3]. His definition was:

```
\makeatletter
\newbox\cut@desc
\newenvironment{cutlines}[1][2]{%
  \@tempcnta=#1\relax
  \setbox\cut@desc\vbox\bgroup
  \parskip=0pt}{%
  \egroup
  \vsplit\cut@desc to \@tempcnta\baselineskip}
\makeatother
```

The argument is the number of lines (default 2).

I tried both of these, and found potential problems with each:

1. The text argument to `\vtruncate` could not include any verbatim material (but this might not be of any concern).
2. If the number of lines specified for the `cutlines` environment was more than the lines in the original text, then the text was padded out with blank lines to make up the specified number.
3. In both cases the final truncated text was not always the specified height, but it was always to within plus or minus a line. However `cutlines` seemed to be more precise than `\vtruncate`.
4. The truncated text ends up in a box that cannot be split across a page boundary.

After some fiddling around[1] I came up with code for a truncate environment that was a mixture of Donald's and Will's code that seemed to avoid the first two of the four problems, and possibly the third as well. The fourth potential problem is inherent in all the proposals.

```
\newsavebox\descbox
\newsavebox\partialbox
\newlength{\vcutl}% for the limit height
\newlength{\Vcutl}% height of full text
\newenvironment{vcutlines}[1][2\baselineskip]{%
  \setlength{\vcutl}{#1}%
  \setbox\descbox\vbox\bgroup
  \parskip=0pt\relax
  }{%
  \egroup
  \Vcutl=\ht\descbox
  \advance\Vcutl \dp\descbox
  \setbox\partialbox\vsplit\descbox to
        \vcutl\relax
  \vtop{\unvbox\partialbox}
  \ifdim \vcutl<\Vcutl \vtruncont \fi}
\newcommand*{\vtruncont}{\noindent\strut\ldots}
```

In the following examples, the test text is:

```
{\itshape
Donald Arseneau created the \verb?\vtruncate?
command and Will Robertson the
\verb?cutlines? environment to truncate text
if it requires more than a specified height.
This is an example, though, of the new
\verb?vcutlines? environment --- a merge
of Donald's and Will's work.}
```

which does include a little verbatim material.

Let's give `vcutlines` a whirl with a limit of 20 lines (i.e., `[20\baselineskip]`).

[1] Quite a lot in fact.

Donald Arseneau created the `\vtruncate` *command and Will Robertson the* `cutlines` *environment to truncate text if it requires more than a specified height. This is an example, though, of the new* `vcutlines` *environment — a merge of Donald's and Will's work.*

And now the same text but with a limit of 3 lines (i.e., `[3\baselineskip]`).

Donald Arseneau created the `\vtruncate` *command and Will Robertson the* `cutlines` *environment to truncate text if it requires more than a spec-*
...

If the text is truncated, as in this example, then the environment finishes by calling the `\vtruncont` macro which by default outputs a final line consisting simply of ... (i.e., `\ldots`) to indicate that the original text continued. A comparison of the height of the original text with the specified height is used to decide if there was truncation.

You can change `\vtruncont` to typeset a different marker, or simply

```
\renewcommand*{\vtruncont}{}
```

to not do anything.

Here's a repeat of the last example:

Donald Arseneau created the `\vtruncate` *command and Will Robertson the* `cutlines` *environment to truncate text if it requires more than a specified height. This is an example, though, of the new*

However eliminating the marker this way seems to lead to a slight problem with the spacing after the end of the environment. Defining instead

```
\renewcommand*{\vtruncont}{\noindent}
```

Donald Arseneau created the `\vtruncate` *command and Will Robertson the* `cutlines` *environment to truncate text if it requires more than a specified height. This is an example, though, of the new*

Gives better spacing after the environment, as shown between this and the example immediately above.

References

[1] Donald Arseneau. `truncate.sty` truncate text to a specified width, 2001. `mirror.ctan.org/macros/latex/contrib/truncate`.

[2] Donald Arseneau. Re: How to limit/cut off text after a number of lines? Post to `comp.text.tex` newsgroup, 16 July 2008.

[3] Will Robertson. Re: How to limit/cut off text after a number of lines? Post to `comp.text.tex` newsgroup, 16 July 2008.

◇ Peter Wilson
12 Sovereign Close
Kenilworth CV8 1SQ, UK
herries dot press (at)
earthlink dot net