Some misunderstood or unknown LaTeX 2$_\varepsilon$ tricks (III)

Luca Merciadri

1 Introduction

After two other *TUGboat* articles, here is a third installment with more tips. We shall see:

1. how to print, in an easy way, a monochrome version of a document,
2. how to draw rightmost braces,
3. how to draw watermarks,
4. a plagiarism discussion and the related computational solutions.

2 Printing monochrome

When writing a ‘screen-version’ of some book, one often uses colors. However, if this screen-version needs to be printed in black and white, it is better to give it as a monochrome document. This can be achieved easily by simply adding `monochrome` to `color` and `xcolor`’s options. For example, if you called these packages without any options, it means that you might put

```
\usepackage[monochrome]{color}
\usepackage[monochrome]{xcolor}
```

in your preamble. Thanks to Enrico Gregorio [3] for this.

Herbert Voß gave me [3] a more technical PostScript version:

```
\AtBeginDocument{%
 \special{ps:
  /setcmykcolor {
    exch 0.11 mul add
    exch 0.59 mul add
    exch 0.3 mul add
    dup 1 gt { pop 1 } if neg 1 add setgray } def
  /setrgbcolor {
    0.11 mul
    exch 0.59 mul add
    exch 0.3 mul add setgray } def
  /sethsbcolor {
    /b exch def /s exch def 6 mul dup cvi dup
    /i exch def sub /f exch def
    /F [ [0 1 f sub 1] [f 0 1] [1 0 1 f sub]
         [1 f 0] [1 f sub 1 0] [0 1 f]
         [0 1 1] ] def
    F i get { s mul neg 1 add b mul} forall
    0.11 mul
    exch 0.59 mul add
    exch 0.3 mul add setgray } def
}}
```

Thanks to him too.

3 Drawing rightmost braces

It is common to synthesize some theorems’ ideas by using a right brace, or simply to write such kinds of systems to define e.g. a function:

$$\left.\begin{array}{ll} -1 & x \leq 0 \\ 1 & x > 0 \end{array}\right\} \overset{\text{def}}{=} f(x). \tag{1}$$

This can be achieved by using various tricks, such as those which were proposed by Eduardo Kalinowski and Dan Luecking [2]:

-
  ```
  \left.
  \begin{array}
  ...\\
  ... \\
  ...
  \end{array}
  \right\}
  ```
- Use the `aligned`, `gathered`, or `alignedat` environments,

but one can also

- define a ‘revert cases’ environment, say `sesac`:

  ```
  \usepackage{amsmath}
  \makeatletter
  \newenvironment{sesac}{%
    \let\@ifnextchar\new@ifnextchar
    \left.%
    \def\arraystretch{1.2}%
    % One might prefer aligns other than left,
    % depending on the use to which it is put:
    \array{@{}l@{\quad}l@{}}%
  }{\endarray\right\}}
  \makeatother
  ```

  in the preamble, the `amsmath` package evidently being mandatory. One can then use `sesac` the way `cases` is used:

  ```
   \begin{equation}
    \begin{sesac}
      -1 & x \leq 0\\
       1 & x > 0
    \end{sesac} = f(x)
  \end{equation}
  ```

  Thanks to Dan Luecking for this [2]. The advantage of this definition is that its call is similar to the one which is used for the `cases` environment. Note that the `rcases` equivalent of `sesac` will be available in the `mathtools` package (from June, 2010), together with variants.

4 Using watermarks

There is sometimes a need for *watermarks*, either for security reasons, or simply for indicating important information along with the document.

There are basically three different ways to put watermarks in your LaTeX 2ε documents:

1. The `xwatermark` package,
2. The Ti*k*Z package,
3. The `draftcopy` package.

We shall discuss these different options separately now. We assume that the user wants to watermark all the pages. (Modifications to make only some pages be watermarked are easy.)

4.1 The `xwatermark` option

To watermark the *current* `tex` document, you can simply put [4]

```
\usepackage[printwatermark=true,
 allpages=true,fontfamily=pag,
 color=gray,grayness=0.9,
 mark=Draft,angle=45,
 fontsize=5cm,
 markwidth=\paperwidth,
 fontseries=b,scale=0.8,
 xcoord=0,ycoord=0]{xwatermark}
```

in your preamble, where parameters are modified in the obvious way.

4.2 The Ti*k*Z way

You can also use Ti*k*Z ([5]):

```
\begin{tikzpicture}[remember picture,overlay]
\node[rotate=0,scale=15,text opacity=0.1]
 at (current page.center) {Draft};
\end{tikzpicture}
```

writes the 'Draft' message in the center of the page, and

```
\begin{tikzpicture}[remember picture,overlay]
\node [xshift=1cm,yshift=1cm]
   at (current page.south west)
   [text width=7cm,fill=red!20,rounded corners,
    above right]
{
This is a draft!
};
\end{tikzpicture}
```

puts 'This is a draft!' in a box, at the desired place. Both might be put outside of the preamble. In both cases, you evidently need to load the `tikz` package. There are many other options (please check the package's manual).

The advantage of this approach is that you can call Ti*k*Z after or before some text to watermark the relative page, without knowing its number.

4.3 The `draftcopy` package

A third approach is to use the `draftcopy` package. You can then specify the intensity of the gray, the range of pages for which the word 'DRAFT' is printed and where it is printed (across the page or at the bottom). The package's feature are best described in its manual [6], but, roughly,

```
\usepackage[english,all,
 portrait,draft]{draftcopy}
```

should suit your needs.

5 LaTeX 2ε and plagiarism

Plagiarism is a well-known issue. It is defined as (*Random House Compact Unabridged Dictionary*, 1995)

> The use or close imitation of the language and thoughts of another author and the representation of them as one's own original work.

I will here take a very concrete case: my own. This year, I had a group project to do, and the two other students did not contribute at all. As I had to share my work with them because there was an oral exam and that the professor wanted it to be shared, I accepted to share it, but with a big watermark.

I had not realized that this choice would be critical. Some days later, I realized that one of the two other students wanted to appropriate the work, and thereby claim its honesty and investment in the work. He tried to remove the watermark, but, despite much research, never found out how. However, he could have done it. I learnt many things thanks to this situation, which I will explain here from a TeX point of view.

My first reaction to ensure security was to *secure the PDF by using free tools*. This was a good dissuasion, as the two other students were stopped by this measure. By 'secure', I mean that I theoretically prevented others from printing, selecting, or extracting content from the PDF file. However, such PDF 'security' is not widely accepted by PDF readers. Here is what Jay Berkenbilt (`qpdf`'s author) once told me [1]:

> The PDF specification allows authors of PDF files to place various restrictions on what you can do with them. These include restricting printing, extracting text and images, reorganizing them, saving form data, or doing various other operations. These flags are nothing more than just a checklist of allowed operations. The PDF consuming application (`evince`, Adobe Reader, etc.) is supposed to honor those restrictions and prevent you from doing operations that the author didn't want you to do.
>
> The PDF specification also provides a mechanism for creating encrypted files. When a PDF file is encrypted, all the strings and stream

data (such as page content) are encrypted with a specific encryption key. This makes it impossible to extract data from without understanding the PDF encryption methods. (You couldn't open it in a text editor and dig for recognizable strings, for example.) The whole encryption method is documented in the specifications and is basically just RC4 for PDF version 1.4 and earlier, which is not a particularly strong encryption method. PDF version 1.5 added 128-bit AESv2 with CBC, which is somewhat stronger. Those details aren't really important though. The point is that you must be able to recover the encryption key to decrypt the file.

Encrypted PDF files always have both a user password and an owner password, either or both of which may be the empty string. The key used to encrypt the data in the PDF is always based on the user password. The owner password is not used at all. In fact, the only thing the owner password can do is to recover the user password. In other words, the user password is stored in the file encrypted by the owner password, and the encryption key is stored in the file encrypted by the user password. That means that it is possible to entirely decrypt a PDF file, and therefore to bypass any restrictions placed on that file, by knowing only the user password. PDF readers are supposed to only allow you to bypass the restrictions if you can correctly supply the owner password, but there's nothing inherent [in] the way PDF files are structured that makes this necessary.

If the user password is set to a non-empty value, neither `qpdf` nor any other application can do anything with the PDF file unless that password is provided. This is because the data in the PDF file is simply not recoverable by any method short of using some kind of brute force attack to discover the encryption key.

The catch is that you can't set restrictions on a PDF file without also encrypting it. This is just because of how the restrictions are stored in the PDF file. (The restrictions are stored with the encryption parameters and are used in the computation of the key from the password.) So if an author wants to place restrictions on a file but still allow anyone to read the file, the author assigns an empty user password and a non-empty owner password. PDF applications are supposed to try the empty string to see if it works as a password, and if it does, not to prompt for a password. *In this case, however, it is up to the application to voluntarily honor any of the restrictions imposed on the file.* [italics mine —lm] This is pretty much unavoidable: the application must be able to fully decrypt the file in order to display it.

None of this is a secret. It's all spelled out in the PDF specification. So encrypting a PDF file without a password is just like encrypting anything else without a password. It may prevent a casual user from doing something with the data, but there's no real security there. Encrypting a PDF file with a password provides pretty good security, but the best security would be provided by just encrypting the file in some other way not related to PDF encryption.

Thus, one might not want to rely only on this PDF feature, especially if the desire is to set attributes without a password (see the slanted sentence in the cited text).

My second idea was to *put a watermark on every page of the document.* For this, I used the `xwatermark` package, because I had no time to look for another way to achieve it (I was near the work's due date).

I then compiled the `tex` document, secured it, and sent it.

In this series of practices, I should have realized that these two protections could totally be circumvented in an easy way. I knew it, partially, but had not much time to think about it. One needs to realize that such practices are not infallible: they are only ways to discourage, not absolutely prevent, your work from being plagiarized.

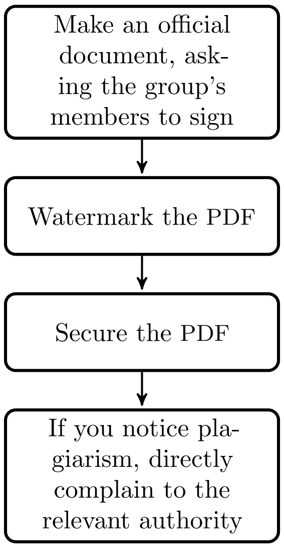
Let's take, for example, the PDF security. One could simply run `pdf2ps`, and then `ps2pdf` on the resulting PostScript file, to recover exactly the same PDF without the security attributes (or hyperlinks). Thus, by using two commands, you can easily remove the PDF protection (assuming it was only concerning attributes, not passwords).

Next, there is the watermark that was $\LaTeX\,2_\varepsilon$-inserted. There are different programs, especially for Windows users, that can remove watermarks. I tried them on my watermark, and none could remove it. Good point, but that does not mean that there is no program which is able to remove it. I might have forgotten one, or simply, a commercial program might be capable of this. (I never test commercial programs.) But I had made an important mistake in my $\LaTeX\,2_\varepsilon$ watermark. The watermark is written on a different PDF 'layer' (one can conceive of a PDF as being constructed from different layers which are superimposed in some way) and is thereby not completely incorporated in the core document. Thus, if you use a LaTeX 2$_\varepsilon$ package to write a watermark in a document, do not forget to *merge layers*. This can be achieved easily. For example, using `lpr` under GNU/Linux, you can re-print the original PDF document (with the different PDF layers) as another PDF where the watermark and the text layers are merged together, thus making differentiating between them very complicated for a 'normal' user. This can be achieved with a GUI too, evidently. For me, I can directly choose this virtual printer, and to print to a file, through GNOME's printing interface.

But one needs to keep in mind that all these measures are here only as a method for discouraging. For example, whatever the protection, one can still take screenshots of the PDF viewer's window, to make copies of the PDF content. This is tedious, but if one wants to, it can be done. If even these screenshots were somehow made impossible, he could use a camera to take pictures of his screen. All the measures you need to take against such behavior need to be adapted, and correlated in regards to other's motivation to exploit your work. This is a very important point, as, once the work is sent, you cannot modify what you gave.

Another point which could be exploited is the use of a somewhat legal document, constraining the group's members to sign.

The best thing is presumably to avoid these problems by talking together, as we humans are equipped with an extraordinary ability to share their feelings and to express themselves; however, communication problems sometimes arise, and, in this case, you might think about the aforementioned tricks. Here is thus what I suggest you to do, if such a problem arises (the first arrow being to follow if communication is somewhat broken):


Make an official document, asking the group's members to sign
Watermark the PDF
Secure the PDF
If you notice plagiarism, directly complain to the relevant authority

◇ Luca Merciadri
University of Liège
`Luca.Merciadri (at) student dot ulg dot ac dot be`
`http://www.student.montefiore.ulg.ac.be/~merciadri/`

References

[1] Berkenbilt, Jay. (PDF specification message), 2010. `http://www.mail-archive.com/debian-user@lists.debian.org/msg570956.html`.

[2] Merciadri, Luca, Kalinowski, Eduardo and Luecking, Dan. 'cases' environment for a brace in the other sense? (`comp.text.tex` discussion), 2010.

[3] Merciadri, Luca, Voß, Herbert and Gregorio, Enrico. dvi, ps, pdf in black and white: how to do it if I have colors? (`comp.text.tex` discussion), 2010.

[4] Musa, Ahmed. The xwatermark Package, 2010. `http://mirror.ctan.org/macros/latex/contrib/xwatermark/xwatermark-guide.pdf`.

[5] Tantau, Till. Ti*k*Z, PGF, 2008. `http://mirror.ctan.org/graphics/pgf/base/doc/generic/pgf/pgfmanual.pdf`.

[6] Vollmer, Jürgen. The draftcopy package, 2006. `http://www.ifi.uio.no/it/latex-links/draftcopy.pdf`.