The Answer Girl BROUGHT TO BOOK

PATRICIA JUNG

The fact that the world of everyday computing, even under Linux, is often good for surprises, is a bit of a truism: Time and again things don't work, or not as they are supposed to. The Answer-Girl in Linux Magazine shows how to deal elegantly with such little problems.

TeX/LaTeX: *A professional text editing environment where the formatting instructions, as with HTML, are written as clear text commands in the ASCII document text and converted by the programs latex or tex respectively according to rules of typesetting. Especially good for scientific texts of pretty much unlimited length, but also suitable for letters or folios.*

dvips: *The dvi-to-PostScript converter creates PostScript code from the dvi ('device independent', hence the name) format generated by TeX/LaTeX. In most cases dvips is preconfigured such that a dvips filename.dvi sends the PostScript code direct to the default printer. With the option -o filename.ps in this case you are forcing dvips to print it instead to the file filename.ps.*

Printouts, each placed, in the West at least, with boring monotony page after page each on an A4 sheet, certainly have their proper place in everyday life. But when you would prefer to honour your beloved with the poetry collection you've written yourself, then you need to come up with at least a nice A5 notebook, printed on both sides.

The paper-saver has less romantic reasons: Printing out the 100 page thesis for proof-reading, on your home PC's two pages per minute inkjet printer, is not good for either the nerves (inks, like paper, are dear, not to mention the time), nor does the resulting mountain of paper provide any particular motivation to revise. Psychologically, a thinner stack is better. So whether it's trees, zeal or nerves which need saving – to accommodate several logical pages on one side of an A4 sheet has to be a good idea.

All in one format

How nice that with PostScript a standard format exists, through which (almost) all printed matter under Linux has to pass: These are passed in the form of a PostScript file from applications to the print machinery, and as such they can also be turned by hand using the *lpr* command into the content of a print job.

Using the infamous Print to file, from every word processing (and also of course from other applications such as Web browsers, the **dvips** used by **LaTeX** scribblers etc.), you obtain a PostScript file which can still be edited before the actual printing.

Anyone who now thinks of PostScript as a programming language, which also has to be put into shape by hand with an editor, is not necessarily wrong, yet a glance into the PostScript code (e.g. with

less filename.ps and a *q* for "*q*uit" to finish, when the pain has receded) ought to make it clear: This is not something which would fit into a single article but would need a lot more space.

So what we need are some tools which take PostScript as an input format, pack several old pages onto one, rearrange them as necessary according to our specifications and at the end spit out PostScript again.

The tool hunt

Let's just see if our clever Manpages have something to offer us here: *man -k* (*-k* for *k*eyword) which is often also found under the name of *apropos*, should at least throw light on the question of whether it is in fact worthwhile tackling this problem with installed on-board resources.

```
[trish@lillegroenn answergirl]$ apropos post➥
script
[...]
mpage (local) - print multiple pages per➥
              sheet on
              PostScript printer
[...]
pstops (1)    - shuffle pages in a PostScript➥
              file
```

Although there is a whole pile of tools involved somehow or other with conversions to and from PostScript, after an introductory reading of the short descriptions there is not that much left – and it doesn't even say that your own distribution and your installation spits out precisely what you see above.

As querying the packet manager shows, *pstops* is hidden in the packet *psutils* (which comes with most distributions, but in case of doubt it can be found at *http://rpmfind.net/linux/rpm2html/search.php?query=psutils*):

```
[trish@lillegroenn answergirl]$ rpm -q -f `w➥
hich pstops`
psutils-1.17-3
```

The "*R*ed Hat *P*acket *M*anager" receives queries ("*q*ueries") with the option *-q* . We tell it what we want to know with *-f*: in this case, to which packet the following specified *F*ile belongs.

Since a file is only unequivocally specified when not only its name (*pstops*), but also the associated directory path is specified, it is not enough simply to say *-f pstops*. In the case of programs existing in the search path you can find out, though, with *which programname*, where they are. If we enclose this command in *Backquotes* (`), then it evaluates the shell first and sets the result (in most distributions */usr/bin/pstops*) into the query command so that we actually ask for *rpm -q -f /usr/bin/pstops*.

mpage on the other hand is usually in a packet of the same name, which can be found in case of doubt at *http://rpmfind.net/linux/rpm2html/search.php?query=mpage*.

Cutting your coat according to your cloth

A brief look at the Manpages for *pstops* and *mpage* promises the solution for ambitious booklet publishers with the former of these tools, but the latter seems easier to use. Since it also appears to be tailored perfectly to the task of 'Cutting stacks of paper down to size by printing several logical pages on one side of A4', we will first let our laziness win.

mpage is often preconfigured such that its output goes direct to the default printer. What is useful later is something we want to avoid right now, since what's the point of saving paper, if on the way there we produce masses of test print-outs? As befits a proper Unix filter program, *mpage* can divert its PostScript output via the arrow > into a file.

With time-served PostScript display programs such as *gv* and *ghostview* or the KDE alternative *kghostview* (which can also be used from the KDE file manager via Drag & Drop) the test file thus produced can always be inspected, before capturing it on paper.

Thus equipped, the first *mpage* exercise is not hard: If one believes the Manpage, *mpage -2 thesis.ps > proofreading.ps* captures the file *thesis.ps* on the virtual sheet such that an odd-numbered and the following even-numbered page (thus a total of *2*) land on one A4 side.

Neatly, *mpage* has drawn a pretty frame around each logical page – but when printed out, this, together with the entire content of the page, is sitting anything but centred on the A4 sheet. There must be something wrong with the format...

In actual fact, *mpage* has the option *-b*, after which the paper format has to be specified without blank spaces. If the brief description of the program available with *mpage -x* declares as follows:

```
-b papersize: A4|Letter|Legal (default US le➥
tter)
```

that the compiled default is the *US letter* format, this offset is no surprise. Then there is nothing for it but to enhance the *mpage* command by *-ba4*, as long as we are printing on A4. Just when you want to capture unnumbered pages on paper, nevertheless, (like those produced for example by Netscape 4.x under *File/Save as...*, when selecting as *Format for saved document* the format *PostScript*), an *-X* helps, which perpetuates the name of the file and the date in the header, when keeping order. The option *-o* as in 'outlines off' switches off the frames created by default around each individual logical page.

All this can of course be used in Netscape etc. even without dumping into a real PostScript file: An *mpage -ba4 -X* | *lpr* as *Print Command* as shown in Figure 1 *File/Print* dialog prints the displayed Web page together with page numbering and with the source **stdin** on A4 paper, where – the option *-4*

stdin: *The standard input, in the shell normally the keyboard. Here in Netscape this statement only shows that the print editing goes, not direct from the file displayed, but via filters, whose output applies as in a Pipe (|) as standard input for the next command*

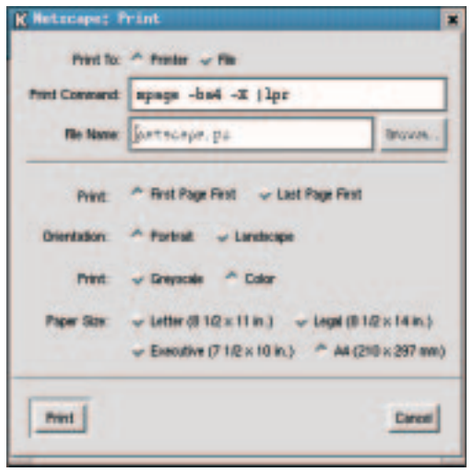


Figure 1: Affecting the print output of Netscape with *mpage*

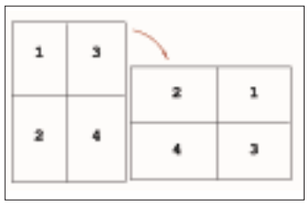


Figure 2: Default arrangement of logical pages with *mpage*: on the right the output material was in landscape format, left in portrait format

Portrait/Landscape: *Upright or horizontal format.*
[]: *In the Backus-Naur notation, in which the Manpages in section Synopsis specify the whole syntax of a command at a stroke, arguments in square brackets are optional.*

does not need to be stated at the same time as default setting – four logical pages land on one page of A4.

These are then arranged as in Figure 2 – left, if Netscape orients the logical pages in **Portrait** format, right in **Landscape**, before *mpage* gets a look in. If, on a page in portrait format, you want to have the logical pages 1 and 2 at the top from left to right and then 3 and 4 underneath, you have to select the *mpage* option *-a*.

The alert Manpage reader has of course already noticed that *mpage*, in addition to other exciting options, also controls one which makes piping the *mpage* output into the print command *lpr* (if necessary at all) superfluous: *mpage -P* prints out on the standard printer on its own (as already mentioned this flag can, if necessary, be left out altogether, depending on the pre-configuration), with *-Pprintername* another installed printer is selected with *lpr*.

Yet with all this abundance of *mpage* options, there is no hiding the fact that disappointment is on the way: booklets cannot be produced with *-E* (every 1st and 4th page together on one page) and *-E* (every 2nd and 3rd page on one page) – unless this is a very thin booklet, consisting of just one sheet of paper.

Order out of Chaos

This is where *pstops* by its very name recommends itself as a panacea in matters of internal PostScript conversion. But not even this tool can perform magic and it expects the source file to comply with the conventions for structuring of PostScript documents (*D*ocument *S*tructuring *C*onventions – DSC) of the PostScript inventor Adobe.

Unfortunately not all PostScript outputting programs do this, nor does *pstops* have control of all DSC requirements from A to Z, so it can happen that the tricks of *pstops* fail.

But that's no excuse. Now is the time to be brave and look the *pstops* Manpage right in the eye. And so that we can see the wood for the trees, it's a good idea to specify just what we want. We would like to print out our poetry collection in such a way that it becomes a booklet. For this to work, the number of logical pages must be divisible by four – *pstops* will add as many blank pages at the end as necessary so it fits. On the front page of a sheet of A4, then, the first (right) and the last (left), the third (right) and the last but two (left) etc. On the back of sheet 1 come page 2 (left) and the penultimate page (right). A 12-page booklet looks like the one in Table 1a.

But since we cannot feed *pstops* with a table like that, we must formalise the whole thing a bit. If we count thoughtfully through the pages to be printed so that we start again at *0* every four pages (so with twelve pages, 0 1 2 3 0 1 2 3 0 1 2 3), the following scheme results: On the first front page to be printed, on the right there is the first, not yet printed, logical page, which was thoughtfully marked with *0* during the count. On the left next to this is placed the last unprinted page, thus the first page from the back, where dividing the number of pages minus 1 by four produces the remainder 3.

On the back page with its back to the last page comes the second page from the back (remainder: 2), on the left next to this is placed the second page from the front (remainder 1). The entire procedure – we divide the number of pages-1 by 4 and write down the remainder – incidentally, this is known as *modulo division*. The first four logical pages are thus crossed off the list, the old page 3 becomes the new logical page 1, and we start again at the beginning with the modulo numbering: 0 1 2 3 0 1 2 3. Here, surprise surprise, the pattern repeats itself: the next front page contains the first page from the back with remainder 3 and the first page from the front with remainder 0. The back page in turn is shared by 1 from the front and 2 from the back.

The same little game is repeated with the remaining four logical pages, and with the pattern thus found (cf. Table 1b) we have found the start of the page print specification, which is stated thus in the *pstops* Manpage:

```
pstops [...] pagespecs [...]
[...]
pagespecs follows the following syntax:
  pagespecs = [modulo:]specs
  specs     = spec[+specs][,specs]
  spec      = [-]pageno[L][R][U][@scale][(x?
off,yoff)]
```

We have the divisor 4 for the *modulo* value, the front pages' *specs* begin with the *spec* for the left page: a -, because we want to have one page from the back, followed by the *pageno 3*. As we still need a *spec* for the right page as well, there is a + and a new *spec*: no minus this time, as we take a logical page from the front, and as *pageno* we had discovered the 0. This means the *pagespecs* for all printed front pages so far read:

```
4:-3+0
```

Similarly a look at Table 1b helps to write out the page description found so far for the print back page:

```
4:1+-2
```

Rotate, move, reduce in size

This is all very nice, but how do we just get two logical A4 pages onto one? Quite simple, you say: they just have to be rotated through 90 degrees, reduced by half and moved into the right place.

Unfortunately we find out from the Manpage on *pstops* that these operations are not always executed in the sequence

1. Move by *xoff* in x-axis and *yoff* in y-axis,
2. 90 degrees rotation anticlockwise (*L* for "*l*eft"), clockwise (*R* – "*r*ight") or 180 degree rotation ("*U*-turn" – *U*) and if necessary
3. scaling to *scale* of the original size. The best thing to do is take a couple of sheets of A4 paper and play around a bit with the geometric operations.

If we lay down the sheet in portrait format (which is the usual way for most printers) after all the transformations, we must land back in this format and at this point. If we decide to turn the sheet anticlockwise by the left bottom corner, then accordingly there is nothing else for the bottom left corner of the future left half of the sheet than to land at bottom right. To do this we must first move the portrait-format sheet horizontally, by a distance equal to its width (Figure 3). Since an A4 sheet is 21cm wide and 29.7cm long, we expand our *pagespecs* for *V=front pages* to

```
4:-3(21cm,0)+0
```

and for *back pages* to

```
4:1(21cm,0)+-2
```

Then we can rotate undisturbed as in Figure 4 round to the left by 90 degrees, which our *pagespecs* expands as follows:

```
4:-3L(21cm,0)+0
4:1L(21cm,0)+-2
```

The left page is now such that all we have to do is reduce it so that it comes to lie in the appropriate place at the right size. A5, the target size, is half as big as A4. But anyone who now simply halves length and width reaches a wrong answer, as it is too small, as Figure 5 shows.

This is not surprising because the halving relates not to the dimensions of the page but to the area. Stupidly, *pstops*, when scaling, is not interested in the ratio of the area and actually wants to see side ratios where different scalings in x- and y-axis are not possible.

A look at Figure 5 or the living paper object luckily shows that the rotated page must be reduced until the longer side corresponds to the old A4 width. The width of an A5 page in turn corresponds to half of the A4 length, thus 29.7/2cm = 14.85cm. If one compares the widths of A5 and A4 (14.85cm : 21cm) or again, the lengths (21cm : 29.7cm) with each other, then one obtains the scaling factor *0.7*. Figure 6 shows that we are spot on with this and can thus expand our page specifications as follows:

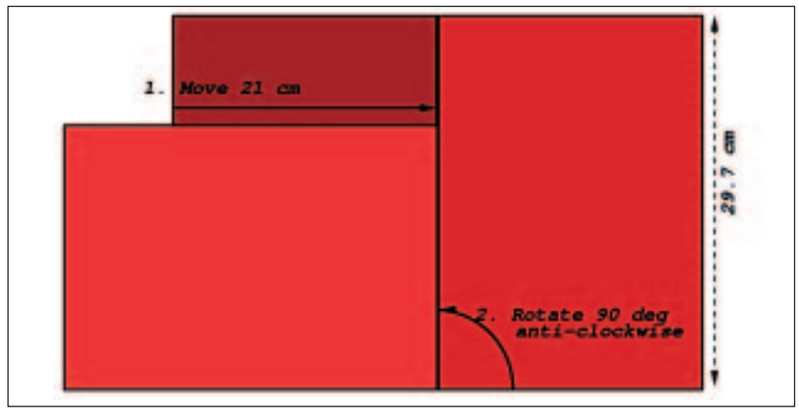


Figure 4: 2nd step: Anticlockwise Rotation

```
4:-3L@0.7(21cm,0)+0
4:1L@0.7(21cm,0)+-2
```

This leaves the right pages: When it comes to rotation and scaling, we can leave everything the same:

```
4:-3L@0.7(21cm,0)+0L@0.7
4:1L@0.7(21cm,0)+-2L@0.7
```

To do this, all we need do is move our A4 sheet a bit differently at the beginning: not just the 21 horizontal centimetres, but also by half the A4 length (thus 14.85cm) upwards (Figure 7).

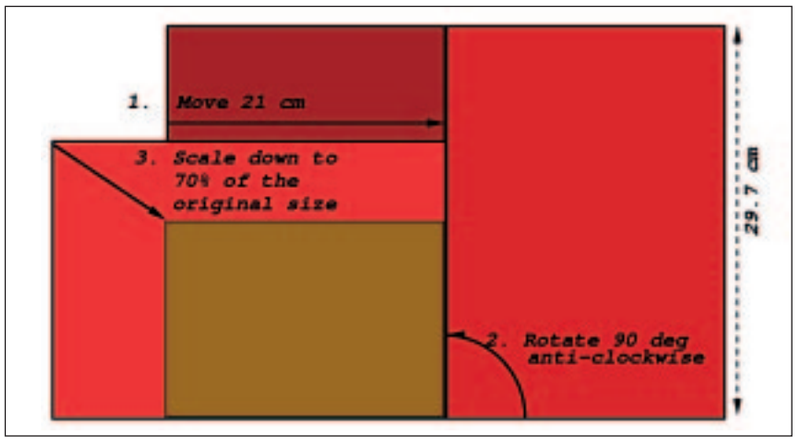
Figure 5: Wrong: Reducing page dimensions by half

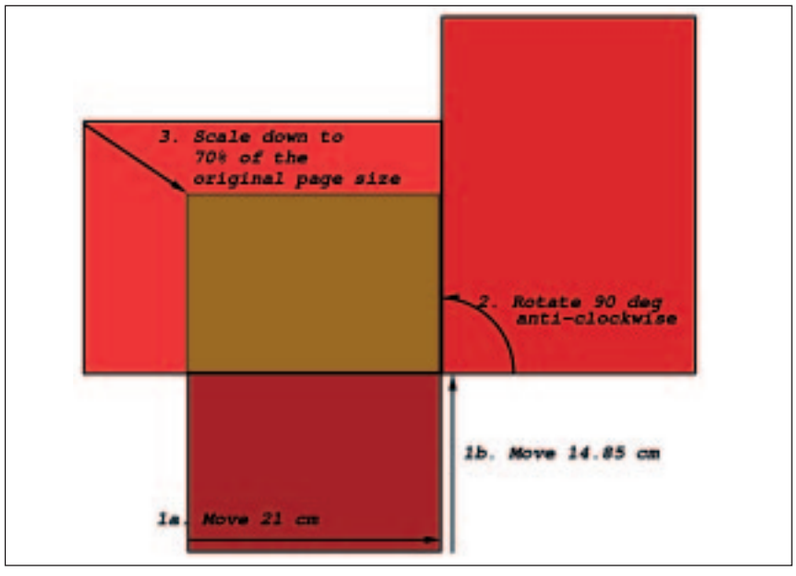
Table 1a: sheet division for a 12-page A5 booklet

A4-sheet No.	Front page		Back page	
	left	right	left	right
1	12	1	2	11
2	10	3	4	9
3	8	5	6	7

Table 1b: ... reduced to a formula

Pages not yet taken into account counted from 0 to 3:	3	0	1	2
From the front (+) or from the back (-)?	-	+	+	-





[top] Figure 6: 3rd step: Reducing the area by half

[above] Figure 7: Steps 1 to 3 for future right page halves

This means our final *pagespecs* look like this:

```
4:-3L@0.7(21cm,0)+0L@0.7(21cm,14.85cm)
4:1L@0.7(21cm,0)+-2L@0.7(21cm,14.85cm)
```

Well printed, tiger

People with printers which automatically print double-sided now have it easy as they can write all the pages at once into a file:

```
pstops -pa4 "4:-3L@0.7(21cm,0)+0L@0.7(21cm,➥
14.85cm),4:1L@0.7(21cm,0)+-2L@0.7(21cm,14.85➥
cm)" poetry.ps poetrybook.ps
```

As page size we specify with *-p* (for "*p*age") *a4*. The error message

```
pstops: page specification error:
 <pagespecs> = [modulo:]<spec>
 <spec>      = [-]pageno[@scale][L|R|U]
               [(xoff,yoff)][,spec|+spec]
               modulo<=1, 0<=pageno<modulo
```

gives us a shock, though, and sends us back to a thorough study of the Manpage. In fact, to combine the specifications for front and back pages into a single *pagespec*, *modulo:* must appear only once at the beginning:

```
pstops -pa4 "4:-3L@0.7(21cm,0)+0L@0.7(21cm,➥
14.85cm),1L@0.7(21cm,0)+-2L@0.7(21cm,14.85cm➥
)" poetry.ps poetrybook.ps
```

The less-fortunate users of single-sided printers generate two files for the front pages...

```
pstops -pa4 "4:-3L@0.7(21cm,0)+0L@0.7(21cm,➥
14.85cm)" poetry.ps poetrybook-front.ps
```

... and the back pages:

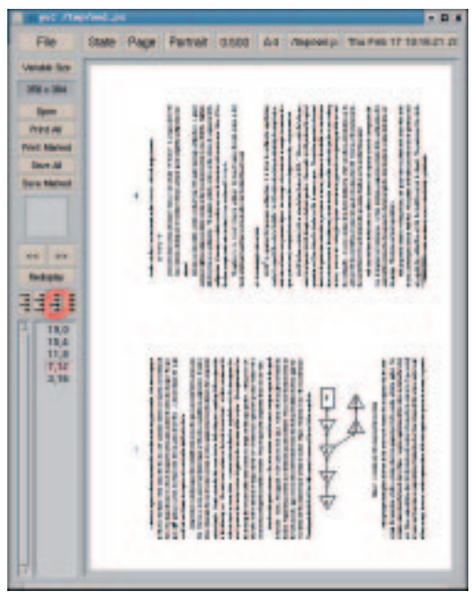
```
pstops -pa4 "4:1L@0.7(21cm,0)+-2L@0.7(21cm,➥
14.85cm)" poetry.ps poetrybook-back.ps
```

... and have to place the stack of paper from the "*lpr poetrybook-front.ps*" print run back in the paper tray. In multi-user environments it is advisable to wait to do this until a time when other users are unlikely to be printing, so that you don't suddenly find a business letter from your colleague on the back of your most passionate love poem.

But even if the timing is right, you still have to find the right position for the sheet of paper – not always a simple task. This undertaking is best approached by loading *poetrybook-front.ps* in *gv* or similar, merely marking the first page (Figure 8) and printing out just this one with *Print Marked*.

If you have not found out how the first page of *poetrybook-back.ps* will be correctly reproduced on the sheet thus printed, you can test the first two pages to find out if the sheets printed with *poetrybook-front.ps* are also actually arranged such that the pages in *poetrybook-back.ps* really do end up on the back of the appropriate front pages.

If at this point you find that the printer spits out the *poetrybook-front.ps* pages so that *poetrybook-back.ps* is better printed with the last back page first, *mpage -r -1 -o* for example may be of assistance. ■



[right] Figure 8:The selected page is marked with the third marking button under *Redisplay* in gv