Werner,

There can be lots of cases when a properly working **cf** could be very helpful. Here's a simple example, an eps file containing a full-page ascii85 encoded image (the middle of standalone-eps.eps is erased and it is converted to pdf in order to comply with the 100k limit).

This file is to be included in a groff document, with a bit of editing (it needs to be shifted up and right and clipped), whilst not changing the original.
Yes, I could edit the file, I could print it separately, I could write a page number on it by hand, I could merge it into the printed groff document, but I think that it is quite legitimate to ask a word processor today to do this donkey work for me.

And why full-page? For simplicity. This way there is no need to deal with the placement of text, that would only cloud the issue. What is the issue? It is the lack of the graphics equivalent of Unix's **cp** command. In Unix analogy we have three variants of **cp** here:

cf	does copy verbatim, can always be used, but it selects its own destination.
trf	can always be used, has the right destination, but sometimes it makes modifications.
file	copies verbatim, to the right destination, but sometimes one can not use it.

Jeez, what would Unix do in such a situation? The nemesis of **trf** is an ascii85 encoded image that may legally contain lots of backslashes. So why not asciihex encoding instead? Because reducing many big images to nearly half size without loss of quality is something to be cherished.

A macro to include a full-page image in a groff document

The macro (pG_eps1) on the following pages has 4 parts. The beige part is fair dinkum troff, parts 2-4 are PostScript. Part-2 is the preparation for the image. It calculates the dimensions of the groff playground, and the viewport for the image. Sets-up clipping in case the image needs to be trimmed.

This is a \Y type embedded auxiliary macro. It would be nice if one could upload the image here and finish the whole business here. However, the file is ascii85 encoded, only \X'ps: can handle that, and \X'ps: would not be welcome within a \Y thing. So you need to close part-2, open part-3 for \X'ps:, then open another auxiliary \Y type macro to finish the business (e.g. plotting the playground boundary, that was not possible whilst in the clipping regime.

In grops' output parts 2-4 are bracketed by the EBEGIN and EEND commands from the prologue. These separate the user's PS and groff's PS. Originally they were not setup correctly, they only prevented the spoiling of grops's PS variables. For better protection they should implement save-restore pairs. That's what I use.

Now a slight problem. Part-1 ends with a routine that would process the image immediately following the routine. Hmmm, there is a restore (to a previously saved state) and save pair between them. The poor image can not meet its processor. Well, this can still work. The P_image_gc1 routine uses readline on currentfile to extract information from the header of the eps file, until it reaches the pixel data. What it does not need, it ignores them. Jeeez, it devours the EEND-EBEGIN pair! Ingenious? Yes. Is this done in a way one could recommend to others? Well, no. What was the problem? That ascii85 files could only be handled with **file** and this construct could not be used within part-2.

Including PS files with no ascii85 images

Easy. No ascii85, the whole PS program can be read into a macro, the macro can be uploaded into part-2, no need for parts 3-4. Thanks God.

1st part of the macro: fair dinkum troff stuff

```
.
.\"-+----- Full page "standard" eps image:
.\"                         size & placement adjustments (left, bot, ...)
.\"                         optional stroking of image bbox and groff playground
.\"
.\"        if debug is set: no clipping
.\"                         plot mm grid
.\"                         stroke image bbox
.\"                         stroke playground
.
.de pG_eps1     \" 1: directory of eps-file: envir var, no $
.               \" 2: name of eps-file, no leading slash
.               \"
.               \" optionals (defaults provided)
.               \"
.               \" 3: "[dx1 .. dy2]" extend playground for mapping (millimetres)
.               \"                   dx1 < 0 and dx2 > 0 is shift to right
.               \"                   dx1 > 0 and dx2 > 0 is making it bigger
.               \"                   subject to clipping to groff playground
.               \" 4: (h)            horizontal adjustment (l) or (m) or (r)
.               \" 5: (v)            vertical   adjustment (b) or (m) or (t)
.               \" 6: 1-4            mapping scheme: 1: full-fill: non-uniform
.               \"                                   4: uniform scales
.               \"                                   3: uniform, keep width
.               \"                                   4: uniform, keep height
.               \" 7: (s)            stroke image bbox
.               \" 8: (s)            stroke groff playground boundary
.
.\"--- parameters into strings and registers
.
.   fullname1 \\$1 \\$2\"               \" assemble full name in string t_full
.
.   ds t_ext \\$3\"
.   ds t_adx \\$4\"
.   ds t_ady \\$5\"
.   nr t_map \\$6\"
.   ds t_img \\$7\"
.   ds t_pgr \\$8\"
.
.\"--- defaults
.
.   if !\\w'\\$3'  .ds t_ext "[0 0 0 0]
.   if !\\w'\\$4'  .ds t_adx (m)
.   if !\\w'\\$5'  .ds t_ady (m)
.   if !\\w'\\$6'  .nr t_map  2
.   if !\\w'\\$7'  .ds t_img (S)
.   if !\\w'\\$8'  .ds t_pgr (s)
.
.\"--- new page, grid if debugging is on
.
.   pG                                  \" new page
.
.   if \\n[debug]=1 .grid_a4            \" draw grid first
.   if \\n[debug]=1 .ds t_img (s)       \" set "stroke image bbox"
.
.\"---
```

EBEGIN

2nd part: PostScript, prepare for the image

```
.
.\"------- 1st auxiliary, ending with image setup (but not the image)
.
.de pG_eps1_aux1 end1
    ps: exec

%--- dimensions of playground (x1 ... y2), viewport of image (X1 ... Y2), MMs

    /x1 \\n[.o]        G_u2mm d_  /x2 \\n[.ll] G_u2mm     x1         add d_
    /y1 \\n[bott_orig] G_u2mm d_  /y2 \\n[.p]  G_u2mm \\n[tm] G_u2mm sub d_

    \\*[t_ext] l~ y2      add /Y3 e_ x2      add /X2 e_ % extend pg for image
                  y1 exch sub /Y2 e_ x1 exch sub /X1 e_

%--- calculate playground path, stroke with red if debug, clip if not

    x1 y1 x2 y2 P_tport_path1                          % clip path

    \\n[debug] 1 eq { .6 P_width P_red S_ } { clip N_ } ifelse

%--- set-up image proc routine: to be followed by the eps file by "\X ..."
%                               extracts Width Height gray/colour hex/85
%                               skips everything until "} exec"
%                               including EEND EBEGIN, why do I have to do this
%
    [[ X1 Y2 X2 Y3 \\*[t_adx] \\*[t_ady] \\n[t_map] \\*[t_img] ]] P_image_gc1

.end1
.
.\"------- upload 1st auxiliary into the PS file
.
\Y[pG_eps1_aux1]
```

EEND
EBEGIN

3rd part: PostScript, upload the image file

```
.
.\"------- upload the image, can be ascii85
.
\X'ps: file \\*[t_full]'
```

EEND
EBEGIN

4th part: PostScript, finish the picture

```
.
.\"------- 2nd auxiliary to finish PS: no clipping here
.\"                                   dimensions to be recalculated (EEND ...)
.\"                                   stroke groff playground if wanted
.
.de pG_eps1_aux2 end2
    ps: exec

    /x1 \\n[.o]        G_u2mm d_  /x2 \\n[.ll] G_u2mm     x1         add d_
    /y1 \\n[bott_orig] G_u2mm d_  /y2 \\n[.p]  G_u2mm \\n[tm] G_u2mm sub d_

    \\n[debug] 1 ne \\*[t_pgr] (s) eq and               % tport frame if wanted

    { x1 y1 x2 y2 P_tport_path1 .3 P_width P_gray_70 S_ } if

.end2
.
.\"------- upload 2nd auxiliary (closed with EEND)
.
\Y[pG_eps1_aux2]

..
```

EEND

I hope that this example makes it clearer why we need a new version of **cf** in groff.

Thanks,

Miklos