Archivos de tipos para TEX

Javier Bezos*

4 de agosto de 1999

Este artículo recopila, con algunas correcciones, una serie de cinco mensajes enviados a la lista de correo Spanish-TeX como respuesta a una pregunta formulada por una de las personas suscritas.

1. Tipos

Esta primera sección está destinada a los diferentes formatos de definición de tipos, es decir, de los archivos que describen la forma de las letras.

MetaFont

El más veterano y, sin embargo, el más refinado y perfecto de todos en cuanto a diseño tipográfico se refiere. Lo creó el propio Knuth y de él han bebido todos. Sus principales inconvenientes son que sólo lo entiende TEX y que es necesario crear previamente los archivos con el dibujo tal y como se envían a la impresora y con la resolución correcta (con el gasto de memoria que implica).

.mf [texto MetaFont] Contiene código de MetaFont que describe con un programa la forma de las letras. A menudo un cierto tipo requiere más de un archivo mf.

.pk [binario] Contiene las letras ya dibujadas (con diferentes resoluciones: .600pk, .300pk, etc. La extensión exacta depende del sistema.)

.gf [binario] Es un paso intermedio de mf → gf → pk y no es esencial.

PostScript

Es el estándar de la industria. Cada tipo en cada una de sus variantes de negrita, cursiva... tiene tan sólo un archivo con el programa para dibujar las letras; esto lo hace la propia impresora sobre la marcha. Las ventajas son su excelente calidad, el ahorro de memoria y la capacidad de ser manipuladas casi sin límites.

.pfa /.pfb [parte texto, parte binario] Contiene el programa ya compilado que usará la impresora. El nombre del archivo no es el del tipo; el nombre que nos «aparece» en el sistema operativo y los programas puede no ser igual y generalmente es más largo y preciso.[1] Los archivos .pfa y .pfb difieren en su forma interna, pero son equivalentes. Hay que consultar el manual de la impresora para saber cuál usar.

*jbezos@arrakis.es. Paquetes disponibles de este autor: accents, polyglot, spanishb, tensind, esindex, titlesec, titletoc.

[1] Se puede consultar el nombre del tipo abriendo el archivo con un procesador como emacs o Alpha; suele estar en la primera línea y, con el añadido de una barra, tras /FontName.

.afm [texto] Informacion legible sobre el tipo. No es necesario, a menos que pensemos meternos bien de lleno en la manipulacion de tipos.

.pfm [binario] Una versión binaria de `afm` específica para Windows.

TrueType

Fue creado por Apple para no depender tecnológicamente de los tipos PostScript de Adobe, pero su calidad resultó ser inferior y terminó en un fiasco. Si MicroSoft no lo hubiera comprado es probable que hoy hubiera desaparecido. Su segundo inconveniente es que hoy sólo se toma en serio en el mundo Windows no profesional, y por ello tan sólo lo aceptan unas pocas implementaciones de TEX, entre ellas, y como notable excepción, pdftex.

.ttf [binario] El equivalente del `.pfa`/`.pfb`.

En sistemas Mac las extensiones en los nombres archivos no son necesarias y, por tanto, no siempre están. Aunque este sistema acepta los tipos tal y como se usan en otros sistema, los PostScript se prefieren en un formato llamado `llwf`.

Ahora llega la primera regla de oro:

Para usar un tipo es necesario tener el archivo con el tipo, ya sea `pk`, `ttf` o `pfa`/`pfb`.

La única excepción es el `pk`, que se puede generar con MetaFont a partir de los `mf`. (MetaFont siempre viene con TEX, salvo en unos pocos casos.) Sin un tipo, simplemente, no es posible seguir.

2. Las métricas

Archivos básicos

Esta parte está dedicada al único archivo que necesita TEX para componer un documento.

TEX no sabe nada de letras: lo único que le interesa es su tamaño para reservarle espacio en la página; su forma, su naturaleza o incluso el sitio (archivo) donde está le resulta por completo indiferente. En cierto modo, cada letra de un documento no es más que una orden que le dice a TEX: «reserva el espacio que el tipo actual indica para mi código [ASCII]».[2]

.tfm [binario] Es la extensión de los archivos que contienen la información del tamaño —junto con alguna otra información— de los caracteres de un tipo. Sin el correspondiente archivo `tfm`, TEX no puede seguir y da un error de que no encuentra las métricas.

.pl [texto] Es el equivalente inteligible para los humanos de un archivo `tfm`. Se puede convertir fácilmente de uno a otro con tftopl y pltotf, pero su contenido puede resultar algo oscuro. Hay una extensión de `.pl` llamada `.vpl` sobre la que volveremos en la proxima sección, pero normalmente no se usan.

.afm [texto] Ya lo vimos en la primera parte: es una descripción al estilo de `.pl` que viene con los tipos PostScript. Con la utilidad afm2tfm se puede crear un `.tfm`.

.mf [texto] Al crear un tipo con MetaFont se crean automáticamente los `tfm` necesarios.

No hay equivalente en TrueType a `pl` y `afm` por lo que o bien se busca un archivo `afm` de un tipo PostScript equivalente, o bien se usa alguna utilidad como ttf2pk, ttf2mf o las comerciales de Y&Y.

[2] Los archivos de tipos con una extensión que empieza con `o` son de una variante de TEX llamada Omega: `.opl`, `.ovf`, `.ofm`, etc.

Archivos de LaTeX

Para crear una orden que cambie de archivo `tfm` es necesario usar `\font\nombre=tipo`; por ejemplo: `\font\rm=Times`, suponiendo que exista un `Times.tfm`. Pero esta forma de definir tipos es muy primitiva y no es conveniente, porque a cada variante (negrita, cursiva, etc.) y tamaño le corresponde un identificador *nombre* distinto.

LaTeX 2$_\varepsilon$ introduce un sistema mucho más completo (y más complejo) que permite seleccionar un tipo por sus diferentes características: familia, figura (*shape*), tamaño, etc.

`.fd` [LaTeX] Contiene las definiciones de una familia, que asigna una cierta métrica a una variante determinada. Naturalmente, hacen falta varias definiciones para cubrir diferentes combinaciones. Tomemos un par de líneas de `ot1ptm.fd` como ejemplo:

```
\DeclareFontShape{OT1}{ptm}{m}{n}{<-> ptmr7t}{}
\DeclareFontShape{OT1}{ptm}{m}{it}{<-> ptmri7i}{}
```

Esto *sólo* define los tipos para que se puedan usar si se quiere, pero no cambia la definición de `\textrm`, `\textit`, etc. De hecho ni siquiera hace falta cargarlos: LaTeX los carga automáticamante cuando hace falta alguno de ellos (en medio de un documento, incluso).

Para más información sobre cómo definir archivos `.fd` se puede consultar `fntguide.tex`. Los archivos `.fdd` cumplen la misma función que los `dtx`: son muchos archivos `.fd` que hay que «desempaquetar» con un `.ins`.

`.sty` [LaTeX] Bueno, no es una extensión para los tipos... Hay algunos paquetes que sirven para el «si, quiero» y que cambian las definiciones de `\textrm`, `\textit`, etc. para que seleccione unos ciertos tipos. El más interesante es times, pero hay tambien un palatino, newcent, helvet, etc. Para definir nuevos paquetes o cambiar los tipos de un documento, de nuevo me remito a `fntguide.tex`.

3. Impresión

Tipos reales

En esta tercera parte hablaré de los archivos necesarios para imprimir un `dvi`. La forma en que esto se implementa varía de un sistema a otro, aunque la mayoría siguen, en líneas generales, las pautas indicadas aquí y que corresponden a dvips. Una excepción notable es Y&Y que no admite los tipos virtuales y recurre a un sistema completamente distinto. Textures también presenta diferencias importantes.

En las dos primeras partes he dicho que para usar un tipo era necesario disponer de un `tfm` con el correspondiente archivo que dibuja las letras (`pk/ttf/pfa/pfb`). A cada archivo del primero le corresponde otro del segundo. En el caso de un `pk`, su nombre y el de su `tfm` coinciden (salvo la extensión, claro). Así, si tenemos `cmr10.tfm`, dvips buscaría el `cmr10.pk` del tamaño adecuado (si es que no encuentra antes uno PS [PostScript]). En cambio, con PS eso no es así, sino que se establece una correspondencia entre el nombre del `tfm` y el del tipo en el archivo `psfonts.map`, como por ejemplo:

```
ptmb8r Times-Bold
```

que establecería que los caracteres de la métrica ptmb8r se imprimen con el tipo PS llamado Times-Bold.

Los pk tienen fijos los caracteres, que ocupan las mismas posiciones en todos los sistemas; los PostScript no los tienen fijos, sino que cada carácter está identificado por un nombre de forma que se le puede asignar un código libremente (mediante un archivo adicional). El dvi sólo contiene información sobre la métrica y el código, pero no sabe a que letra corresponde ese código: a un mismo código le corresponden diferentes letras en diferentes sistemas. Por ejemplo, el código 150 es el acento breve en Unix, semirraya (*endash*) en Windows y eñe en Mac. ¡Imaginaos el lío que se puede montar!

Aprovechando que los códigos en PS se pueden asignar libremente, dvips los distribuye de una forma próxima a Windows, no por creer que sea mejor, sino porque Windows depende de TrueType, que tiene ciertas limitaciones. En realidad, la línea completa de psfonts.map de arriba es:

```
ptmb8r Times-Bold "TeXBase1Encoding ReEncodeFont" <8r.enc
```

El archivo 8r.enc establece que el código 150 corresponde al carácter llamado endash. En otras palabras, cuando en un dvi aparece la métrica ptmb8r, dvips usa el tipo Times-Bold con los caracteres distribuidos como indica 8r.enc. (Nota. Cada plataforma viene con una distribución preestablecida, que es la que usan otros programas como Word, WP, etc. Pero con PS se puede cambiar, e incluso acceder a más de 256 caracteres. Con TrueType eso no es posible y si el archivo contiene más de 256 letras, no todas son accesibles.)

Tipos virtuales

Hasta ahora hemos hablado de tipos PS, ttf y pk, que contienen la descripción de la forma de las letras. Sin embargo, hay otra posibilidad más llamada tipo virtual y que no contiene ninguna descripcion del «dibujo», sino que «parasita» otros tipos «de verdad». Esto se consigue mediante un nuevo archivo:

.vf [binario] Contiene información sobre cuál es el tipo que realmente se ha de usar para cada una de las letras.

.vpl [texto] Es el equivalente al .pl de la anterior sección, pero con un cambio importante: .vp normalmente lo obtenemos a partir de un tipo del que hemos extraido sus métricas. Con .vpl normalmente seguimos el proceso contrario: creamos un tipo a nuestra medida escribiendo un .vpl y generamos el .vf con la utilidad vptovf. Estos procesos sólo se utilizarán cuando se desee manipular los tipos y no por el usuario medio. Una utilidad escrita en TEX llamada fontinst es de mucha ayuda en estas tareas, si se piensa investigar más. Otra utilidad es afm2tfm, más rápida y sencilla de usar pero que no genera los fd, algo que, por otra parte, no es complicado hacer a mano.

Normalmente, los vf se emplean para reorganizar las letras siguiendo varios sistemas, pero se pueden utilizar para otros fines como simular versalitas a partir de mayúsculas o crear caracteres que no existen.

Cuando dvips se encuentra en el dvi con un tfm de un tipo del que hay un vf, automáticamente lo sustituye por el tfm que ese vf indique y vuelve a empezar. Si no lo encuentra, pasa a mirar si hay una entrada para ese tfm en psfonts.map y si por tanto hay un tipo PS. Y si esto último tambien falla, ya busca un tipo pk. Por ejemplo, con ptmr8t.tfm busca un vf; lo encuentra y éste reenvía los caracteres al ptmr8r.tfm. Para este último busca de nuevo un vf pero como no lo hay, busca en psfonts.map y encuentra un tipo «real» ptmr8r, que llama al PS Times-Roman.

¿Y qué hay sobre los visores de dvi? Normalmente, funcionan de igual modo que dvips, salvo que no suelen entender psfonts.map (no tienen intérprete PS) y en consecuencia usan alguna alternativa. Como los visores se desarrollan para una plataforma específica, ya conocen el código

que corresponde a cada letra, y por tanto, en lugar de establecerse una correspondencia de nombre a código, se hace directamente de código a código. En todo caso, hay que recalcar que los procesos de visión e impresión *son distintos* y por tanto pueden y suelen requerir archivos distintos. El manual de cada implementación ya explica cómo configurar el visor de dvi.

4. Nombres y codificaciones

En la anterior entrega decíamos que dvips se encarga de convertir el nombre usado por TEX (digamos `ptmb8r.tfm`) en el real del tipo. Pero ¿qué significa el nombre tan obtuso usado por TEX? ¿Por qué no usar directamente el nombre de Times-Bold? La razón es, ante todo, compatibilidad. Con este método nos aseguramos que un tipo usado en un documento se identifica por igual en cualquier sistema. Por ejemplo, Textures identifica Times-Bold como `TimesB.tfm` y los `dvi` asi creados son ilegibles en otras implementaciones.

El nombre procede de una serie de convenciones que permiten identificar unívocamente un tipo con tan sólo ocho caracteres (por culpa de nuestro viejo conocido MS-DOS). La primera letra es el propietario del tipo: `p` (Adobe), `e` (Apple), `l` (Linotype), `i` (ITC), etc.

Las dos siguientes identifican la familia: `hv` (Helvetica), `bk` (Bookman), `ge` (Geneva), `ur` (Centaur), `ag` (Avant Garde), `zc` (ZapfChancery), `tm` (Times), `ut` (Utopia), `cr` (Courier), `pl` (Palatino), etc.

La cuarta letra siempre es el grueso: `r` (texto), `b` (negrita).

A continuacion vienen las «variantes»: `i` (cursiva), `c` (versalitas), etc. Sigue la codificación, que explico inmediatamente. Si ninguna de estas variantes es de aplicación en un tipo y la codificación es desconocida o no estándar, se añade una `r` de relleno (¿tal vez un *horror vacui* de quien ideó estos nombres?). Finalmente, se añade el ancho, aunque si es el normal no se añade nada: `n` (estrecha), `x` (ancha), etc. Si el nombre resultante termina en `rr` se simplifica a una.

Con este sumario y consultando algunos archivos `.fd` se puede entender algo mejor su significado. Por ejemplo, `phvrrn` es Helvetica de Adobe, texto, codificación desconocida y estrecha; `pplri7tx` es Palatino de Adobe, texto, cursiva, con codificación 7t y ancha; las cuatro variantes básicas de Times (`T1`) son `ptmr8t`, `ptmri8t`, `ptmb8t` y `ptmbi8t`. Como este esquema no parecía todavía lo bastante complicado, en algunos tipos se usan otras letras: por ejemplo, las variantes equivalentes de Avant Garde son `pagk8t`, `pagko8t`, `pagd8t` y `pagdo8t`, respectivamente, sin que haya ningun tipo de regla que nos ayude.[3]

Ahora voy a explicar con un poco más de detalle el identificador de codificación con ejemplos concretos para Times-Bold.[4]

8r Es el tipo básico. Por ejemplo, ptmr8r es el tipo real que tiene su correspondiente archivo PostScript con las letras organizadas como ya he dicho. Así pues, los archivos asociados a esta codificación para Times-Bold son `8rptm.fd`, `ptmb8r.tfm` y `Times-Bold` (o como se llame el archivo PS en ese sistema, con extension `.pfa` o `.pfb` salvo en MacOS). No se debe usar directamente, y naturalmente no hay ningún `vf` ya que el tipo es real.

7t Es derivado y solo tiene los caracteres con código menor que 128, distribuidos como se dice en el *TeXbook*. Dentro de LATEX se le conoce como `OT1`. Sus archivos son `ot1ptm.fd`, `ptmb7t.tfm` y `ptmb7t.vf`. Con él, los acentos los fabrica TEX superponiendo el acento y la letra.

[3] Confiemos en que, en un futuro, este sistema de nombres se descarte —poco probable— o al menos se simplifique, porque es una auténtico caos.

[4] Las equivalencias entre estos códigos y los usados por LATEX no son completas, ya que se refieren a cuestiones distintas. Las de LATEX no saben nada de ligaduras, que se crean desde dentro de los tipos; las de los tipos no saben nada de componer acentos, que se combinan en TEX.

8t Es derivado y tiene todos los caracteres distribuidos según se acordó en una convención de TeX celebrada en la ciudad de Cork con los códigos mayores de 127 ocupados por letras con diacríticos. Dentro de LaTeX se le conoce como `T1`. Sus archivos con `t1ptm.fd`, `ptmb8t.tfm` y `ptmb8t.vf`. Con él, los acentos y otras letras para un buen número de lenguas existen realmente en el tipo. Entre ellas, en contra de lo que se dice, *no* está el catalán (falta la ele geminada).

8c Pero si 8t (= `T1`) sólo tiene letras con diacríticos, ¿adónde van a parar los símbolos que normalmente se encuentran en esas posiciones, como tanto por mil, letras voladitas, barra vertical, etc.? 8c se dedica a esos símbolos, junto con otros creados a partir de ellos con la ayuda de un `vpl`. Dentro de LaTeX se conoce como `TS1`. Sus archivos son `ts1ptm.fd`, `ptmb8c.tfm` y `ptmb8c.vf`. Se deben usar a través del paquete textcomp.

Nótese que hay un `fd` distinto para cada codificación/familia y que estas convenciones se aplican a tipos PostScript. Los tipos EC, entre otros, siguen otro criterio.

Para usar un cierto tipo podemos utilizar las herramientas descritas en secciones anteriores y hacer el trabajo nosotros mismos; pero hay una serie larga de archivos bajo el nombre PSNFSS que nos proporciona los más importantes. Para otros tipos, o bien se busca por la CTAN, o bien tenemos que hacerlos nosotros con la ayuda de fontinst.[5] Si queremos usar, digamos, Garamond, el camino más corto es buscar todos los archivos que contengan `pgm` y luego añadir entradas en `psfonts.map` tomando las de otros tipos como modelo.

5. La interfaz de LaTeX

`Inputenc.sty, latin1.def, ansinew.def, applemac.def`, etc.

En una anterior sección decía que cada letra es, en cierto modo, una orden que dice a TeX: «Reserva el espacio que la actual métrica indica para mi código». En realidad, TeX permite asignar a cada letra una definición arbitraria. El paquete inputenc aprovecha eso para definir á —o más exactamente el código que corresponde a la á— como `\'a`; eso es todo lo que hace. Gracias a que ha sido pensado para alguna plataforma específica, el procesador que se usa para escribir el documento establece la equivalencia $225 \leftrightarrow$ *á* y por eso lo vemos en la pantalla. Pero TeX no sabe nada de eso, y por tanto, inputenc se encarga de ello con la línea

```
\DeclareInputText{225}{\@tabacckludge'a}
```

en el archivo `latin1.def`. (Nota: `\@tabacckudge'` = `\'`.) En un Mac, ese código corresponde a un punto centrado y por tanto en `applemac.def` esa línea es:

```
\DeclareInputText{225}{\textperiodcentered}
```

mientras que hay otra que es:

```
\DeclareInputText{135}{\@tabacckludge'a}
```

La orden

```
\usepackage[latin1]{inputenc}
```

carga inputenc y el archivo de definiciones `latin1.def`. (Para Windows 9x es `ansinew.def`.)

Una vez hechas estas definiciones, TeX no necesita saber más sobre la plataforma.

[5] Puede ser interesante consultar los guiones de Perl VFInst.

`Fontenc.sty, ot1enc.def, t1enc.def, ts1enc.def`, etc.

Realiza el proceso inverso: establece la correspondencia entre *á* y el código correspondiente en la codificación usada, que a su vez depende del `tfm` usado. Así, `t1enc.def` incluye una línea que dice

```
\DeclareTextComposite{\'}{T1}{a}{225}
```

porque esa es la posicion que ocupa el «dibujo>"de la letra *á* en los archivos con la codificación `T1`, como ptmr8t (recuerdese que 8t en tipos = `T1` en LaTeX), phvr8t, pcrr8t, etc. (Y los de la familia EC, aunque 8r no este en su nombre.) En este ejemplo concreto, *á* se encuentra en la codificación 8t en la misma posición que en `latin1`, ya que la distribución de Cork es una modificación de la ISO Latin 1.

En cambio, `ot1enc.def` *no* incluye una línea equivalente y no establece ninguna correspondecia. Eso indica que el dibujo para esa letra no está disponible en los tipos con esa codificación (ptmr7t, phvr7t... y los de la familia cm), y en consecuencia hace que LaTeX construya la *á* con el acento por un lado y la letra por otro, por medio de una orden primitiva de TeX.

6. Vida y obra de una *á*

En esta ultima entrega veremos qué ocurre con el siguiente documento desde que lo componemos hasta que lo imprimimos:

```
\documentclass{article}
\usepackage[latin1]{inputenc}
\usepackage[t1]{fontenc}
\usepackage{times}
\begin{document}
\textbf{á}
\end{document}
```

Sobre times ya he hablado en la segunda entrega, y dice (ojo, solo dice) que la familia para la letra de palo cruzado (*serif*) es `ptm`.

Ya podemos decir qué ocurre cuando se alcanza la orden

```
\textbf{á}
```

`\textbf` establece que el grueso de la letra es negrita y le dice a TeX que las siguientes letras del dvi son de la métrica que algun `.fd` asigna a la combinacion `T1/ptm/b/n`, donde `n` es la figura (`n` es normal, `it` es cursiva, `sc` es versalitas, etc.) Los `.fd` se nombran con el par codificación/familia y por tanto LaTeX busca y lee el archivo `t1ptm.fd`.

En éste se encuentra la línea

```
\DeclareFontShape{T1}{ptm}{b}{n}{<-> ptmb8t}{}
```

Así pues, LaTeX ya sabe que la métrica es `ptmb8t.tfm`. TeX carga los datos de ptmb8t en su memoria y escribe en el `dvi` una orden de que las próximas letras serán de ese tipo.

Luego llega la `á` que es una orden definida como `\'{a}`. A su vez `\'` es una orden que toma la `a` e investiga si existe en la codificación actual (`T1`) un código para `\'a`. Existe y es 225, así que TeX reserva en el `dvi` el espacio que `ptmb8t.tfm` indica para el código 225.

El documento ya se termina de componer y queremos imprimirlo. Dvips va leyendo el `dvi` y se encuentra con el cambio de métrica a ptmb8t. Primero averigua si hay un `ptmb8t.vf`; lo hay y dvips memoriza toda su información para más tarde. Luego pasa a investigar qué sigue.

Lo siguiente que encuentra es un carácter con código 225. Busca ese carácter en la información que ha memorizado, y encuentra que ha sido «parasitado» y en realidad es el carácter 225 de ptmb8r. (De nuevo coincide el código, pero no tiene por que ser así.) Como no hay un `ptmb8r.vf`, busca en `psfonts.map` y encuentra que es el tipo real `Times-Bold`, así que escribe al archivo PostScript que se está generando una orden para cambiar a ese tipo y añadir la letra con código 225.[6]

Ya hemos generado el `ps`, que vamos a imprimir. Como ptmb8r sigue la codificación del archivo `8r.enc`, la impresora o Ghostview determina que 225 es la letra llamada `aacute`. Todo lo que queda por hacer es tomar los datos del dibujo de `aacute` en `Times-Bold` e intruir a la impresora para que haga el dibujo físico. (A menudo, el proceso de crear una archivo PostScript e imprimirlo se hace automáticamente en la trastienda.)

7. Tipos en cinco minutos

Los buenos observadores habrán notado que este documento tiene una versión algo más estrecha de Courier. Se puede crear una variante con cierta rapidez si no exigimos rigor, ni pensamos en distribuir los `tfm` creados, ni usamos otros signos especiales que no sean vocales con acento o eñes. Para ello, simplemente copié los `afm` que hay en CTAN y usé afm2tfm del siguiente modo:

```
afm2tfm pcrr8a.afm -e .8 -p 8r.enc pcrr8rn.tfm
afm2tfm pcrb8a.afm -e .8 -p 8r.enc pcrb8rn.tfm
```

Añadí a `psfonts.map` lo siguiente:

```
pcrr8rn Courier " .8 ExtendFont TeXBase1Encoding ReEncodeFont " <8r.enc
pcrb8rn Courier-Bold " .8 ExtendFont TeXBase1Encoding ReEncodeFont " <8r.enc
```

y al archivo de mi visor dvi (OzTeX) donde están los tipos:

```
pcrr8rn   -   "Courier"   Mac8r.enc
pcrb8rn   -   "Courier"   Mac8r.enc  b
```

Gracias a la feliz circunstancia de que las letras con acentos están en las mismas posiciones en 8r y 8t, me limité a copiar entero el archivo `t1pcr.def` en el preámbulo y cambiar lo siguiente:

```
\DeclareFontShape{T1}{pcr}{m}{n}{
   <-> pcrr8rn
}{}
\DeclareFontShape{T1}{pcr}{b}{n}{
   <-> pcrb8rn
}{}
```

Una auténtica chapuza, pero que funciona muy bien en el español. Ojalá la instalación completa y rigurosa no fuera, ni tan siquiera, así de complicado.

[6] El orden real con el que dvips funciona no es exactamente el que digo, pero como si lo «seriese».