

En `LaTeX`, un espacio en blanco en el texto fuente produce un espacio en blanco en el documento compilado. Más de un espacio en blanco en el texto fuente no producen más que un espacio en blanco en el texto compilado.

Por tanto, si escribimos:

```
Uno o más espacios equivalen a un solo espacio en blanco
Uno o más  espacios equivalen a un    solo espacio    en blanco
```

obtenemos en ambos casos: Uno o más espacios equivalen a un solo espacio en blanco.

Podemos usar también el comando `\hspace{'valor'}` para obtener un espacio horizontal igual al *valor* que especifiquemos. Por ejemplo,

```
Hola\hspace{4cm}adiós
```

dejará un espacio horizontal de 4 centímetros entre las palabras "Hola" y "adiós" en el texto compilado.

Sumario

[ocultar]

- [1Espacio de no separación](#)
- [2Sangría de primera línea y espacio entre párrafos](#)
- [3Espacios tras punto](#)
- [4Espacios con extensión infinita](#)
- [5Blancos verticales](#)
- [6Referencias](#)

Espacio de no separación[\[editar\]](#)

`\nonbreakablehyphen` justifica de manera automática los párrafos, por lo que una vez que se llene una línea mandará lo que sigue a la línea de abajo, separando palabras que, en ocasiones, sería mejor

mantener juntas. Para conseguir que `\nonbreakablehyphen` no separe dos palabras con un cambio de línea debemos usar el comando `\hyphen`. Por ejemplo, debemos escribir

```
O.\hyphenWilde escribió obras como...
```

para que no haya un salto entre "O." y "Wilde".

Sangría de primera línea y espacio entre párrafos[\[editar\]](#)

Para ajustar la sangría de la primera línea, hay que modificar el parámetro `\parindent`. Por ejemplo, se deja un cuadratín con:

```
\setlength{\parindent}{1em}
```

Y se suprime con:

```
\setlength{\parindent}{0pt}
```

El espacio entre párrafos se ajusta con `\parskip`:

```
\setlength{\parskip}{10pt}
```

Espacios tras punto[\[editar\]](#)

A menos que se use babel con la opción spanish, de manera automática `\par` deja un espacio adicional después de un punto, según la tradición tipográfica anglosajona,^[1] a menos que éste

esté precedido por una mayúscula, caso en el cual `\par` interpreta el punto como el de una abreviatura y no deja ningún espacio adicional. Si una abreviatura termina con una letra minúscula, como por ejemplo la abreviatura latina "e. g.", entonces hemos de evitar el espacio

adicional que dejará `\par`. Esto se consigue con el comando `\@`. Por ejemplo, debemos escribir

```
...existen clases (e.g.\@ la clase de todos los conjuntos) que no son conjuntos
```

Si una frase termina con mayúscula, `\par`, como ya hemos dicho, no dejará un espacio adicional después del punto que termina dicha frase por considerarlo el de una abreviatura.

Para indicarle a `\par` que se trata efectivamente del punto que termina una frase debemos escribir el comando `\@`. Por ejemplo, debemos escribir

```
Podemos compilar nuestros documentos en formato PDF\@. Además, están los formatos...
```

Espacios con extensión infinita[\[editar\]](#)

Otra opción más para espacios horizontales son los comandos que "empujan" el texto hasta el final de la página. Por ejemplo, el comando `\hfill` empuja el texto dejando espacios en blanco, como en el siguiente ejemplo: Desde este punto\hfill hasta este punto produce:

Desde este punto\hfill hasta este otro

Si en lugar de `\hfill` escribimos `\hrulefill` o `\dotfill` obtenemos, respectivamente:

Desde este punto\hrulefill hasta este otro

y

Desde este punto\dotfill hasta este otro

Blancos verticales^{[[editar](#)]}

Por otra parte, una o más líneas en blanco en el texto fuente producen una sólo línea en blanco en el texto compilado.

Así, si escribimos:

Primera línea.

Aún estamos en la misma línea. Esta es la
segunda línea Esta es la tercera línea

obtenemos:

Primera línea. Aún estamos
en la misma línea.

Esta es la segunda línea

Esta es la tercera línea.

Podemos usar también el comando `\vspace{'valor'}`, con un efecto similar al de `\hspace{}` salvo que el espacio es vertical. Para espacios verticales predefinidos, podemos usar los comandos

`\smallskip` `\medskip` `\bigskip`

Es posible también dejar cierto blanco cuando se hace un salto de línea con `\` añadiendo una dimensión entre paréntesis: `\['valor']` produce un espacio entre líneas igual al *valor* especificado. Recuérdese que esta orden no es para crear espacios entre párrafos, sino solo cuando para los casos, no muy habituales, en los que hay un salto de línea dentro de un párrafo. De hecho, su uso para aumentar el espacio entre párrafos es un error (habitual).

Referencias^{[[editar](#)]}

1. [Volver arriba](#) Sin embargo, el paquete `babel` en su opción castellana sigue la tradición tipográfica europea de no usar espacios extra después de un punto. Por tanto, ahorra todos estos problemas con las abreviaturas.

Manual de LaTeX/Escribiendo texto/Tablas

En los libros escolares, las tablas son normalmente utilizadas para recapitular los resultados de una investigación. En general es necesario manejarlas bien para realizar documentos de buena calidad.

La gestión de tablas no es muy intuitiva. Las tablas de base son fáciles y presentables, utilizando la misma lógica que en HTML, pero una tabla un poco más elaborada requiere de cierto aprendizaje ya que no es muy intuitiva su construcción.

Sumario

[ocultar]

- 1El entorno *tabular*
- 2Tabla de base
- 3Texto en las tablas
- 4El entorno *tabular**, control de la anchura de una tabla

El entorno *tabular*[\[editar\]](#)

Recordemos algunos conceptos ya explícitos.

Entorno

Un entorno es una declaración particular destinada a la composición de texto en un estilo específico. Todos los entornos empiezan y terminan de la misma manera:

```
\begin{nombre-entorno}  
...  
...  
\end{nombre-entorno}
```

Entorno `tabular`

El entorno `tabular` es otro tipo de entorno, concebido para colocar los datos en las tablas. Ciertos parámetros son necesarios después de la declaración del entorno para describir la alineación de cada columna. No es necesario indicar el número de columnas porque se deduce a partir de los parámetros introducidos. De la misma manera, se pueden introducir líneas verticales entre columnas. Los símbolos siguientes están disponibles para describir las columnas de una tabla.

- `l` : Columna alineada a la izquierda
- `c` : Columna centrada
- `r` : Columna alineada a la derecha
- `p{anchura}` : Columna de anchura fija, justificada y con sangría; El texto está posicionado en lo alto de la celda.
- `m{anchura}` : Como en el caso anterior pero el texto está centrado verticalmente.
- `b{anchura}` : Como en el caso anterior, pero el texto está posicionado en la parte baja de la celda.

<cite id="endnote_ los parámetros `m` y `b` necesitan la utilización de la

extensión `array`" style="font-style: normal;">>[[#ref_ los parámetros `m` y `b` necesitan la utilización de la extensión `array`|^]]

- `|` : línea vertical
- `||` : doble línea vertical

Una vez en el entorno,

- `&` : Separador de columna.
- `\\` : Principio de una nueva línea.
- `\hline` : Línea horizontal.

A tener en cuenta, que los espacios insertados entre estos comandos son inútiles, pero facilitan la lectura.

Tabla de base[\[editar\]](#)

Este ejemplo muestra como crear una simple tabla en LaTeX. Es una tabla tres por tres, pero sin ninguna línea.

```
\begin{tabular}{ l c r }
  1 & 2 & 3 \\
  4 & 5 & 6 \\
  7 & 8 & 9 \\
\end{tabular}
```

Modificando el ejemplo anterior añadiendo algunas líneas verticales:

```
\begin{tabular}{ l | c || r | }
  1 & 2 & 3 \\
  4 & 5 & 6 \\
  7 & 8 & 9 \\
\end{tabular}
```

1	2	3
4	5	6
7	8	9

Para añadir las líneas horizontales superiores e inferiores:

```

\begin{tabular}{l | c || r | }
\hline
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\hline
\end{tabular}

```

1	2	3
4	5	6
7	8	9

Y finalmente, para añadir líneas centradas entre todas las filas (ver la utilización del entorno `center`):

```

\begin{center}
\begin{tabular}{l | c || r | }
\hline
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\hline
\end{tabular}
\end{center}

```

1	2	3
4	5	6
7	8	9

Texto en las tablas[\[editar\]](#)

Los algoritmos de LaTeX para generar las tablas tienen ciertas imperfecciones. Una de ellas es que no hará un salto de línea dentro de una celda, aunque se desborde la anchura de la página. Para las columnas que contendrán una cierta cantidad de texto, se recomienda emplear el atributo `p` e indicar la anchura deseada de la columna (aunque esto pueda obligar a efectuar varios ajustes antes de obtener el resultado previsto).

Antes de continuar, tenemos que presentar el sistema de medidas que LaTeX emplea. Es muy flexible para que se pueda elegir entre toda una variedad de unidades de medida

- `pt` : punto anglosajón, 1/72 de pulgada ;
- `mm` : milímetro ;
- `cm` : centímetro ;

- `in` : pulgada (2,54 cm) ;
- `ex` : altura d'x , altura de una letra sin el trazo vertical ni el palo inferior de la fuente utilizada;
- `em` : cuadratín, grosso modo la anchura de una **M** (capital) en la fuente utilizada.

Existen comandos conocidos con el nombre de *commandos de longitud*, que juegan el rol de variable, que no tienen valores fijos porque dependen de la configuración de la clase y/o del preámbulo normal del documento. Los mas útiles son:

- `\parindent` : El tamaño del desplazamiento a la derecha ;
- `\baselineskip` : Distancia vertical entre las líneas ;
- `\parskip` : Espacio suplementario entre los párrafos ;
- `\textwidth` : La anchura de una línea de texto en el entorno local (por ejemplo, las líneas son generalmente mas estrechas en el resumen que en el texto normal);
- `\textheight` : La altura del texto en la página;

Los ejemplos que se dan a continuación son bastante largos debido a que se ilustran lo que se produce cuando hay un fragmento de texto en las celdas de una tabla. Así, en lugar de reproducirlo en la página, id a [\(1\)](#) para poder consultar directamente el fichero LaTeX de ejemplo, [tutorial4/wrapped.tex wrapped.tex] y luego mirar el [tutorial4/wrapped.pdf resultado].

El entorno `tabular*`, control de la anchura de una tabla[\[editar\]](#)

Es fundamental una pequeña extensión de la versión básica de la tabla, ya que exige un parámetro suplementario (antes de las descripciones de columnas) para indicar la anchura deseada para la tabla.

```
\begin{tabular*}{0.75\textwidth}{ | c | c | c | r | }
\hline
label 1 & label 2 & label 3 & label 4 \\
\hline
item 1 & item 2 & item 3 & item 4 \\
\hline
\end{tabular*}
```

Sin embargo, esto no se parece a lo que se espera. Las columnas tienen siempre su anchura normal (justo lo suficientemente larga para adaptar su contenido mientras que las líneas son tan anchas como la anchura deseada de la tabla) La tabla no tiene una buena apariencia. La razón de este desorden es debido a que se tiene que insertar un espacio suplementario en la columna. Latex, tiene una longitud en caucho, que a diferencia de otras, no son fijas y Latex puede dinámicamente decidir el momento en el que deben ser fijas. Así, la solución al problema propuesto es:

```

\begin{tabular*}{0.75\textwidth}{@{\extracolsep{\fill}} | c
| c | c | r | }
\hline
label 1 & label 2 & label 3 & label 4 \\
\hline
item 1 & item 2 & item 3 & item 4 \\
\hline
\end{tabular*}

```

En el código se ha introducido la construcción `@{...}` que se coloca al principio de la columna. Mas tarde se darán los detalles de este elemento. En el interior de estas construcciones, el comando `\extracolsep`, exige una anchura como parámetro. Se hubria podido utilizar una anchura fija, sin embargo, utilizando una longitud elástica, es decir, `\fill`, las columnas se espacian automáticamente de manera uniforme.

Manual de LaTeX/Listados de código/Listados con listings

< [Manual de LaTeX](#) | [Listados de código](#)

Sumario

[ocultar]

- 1El paquete *listings*, modo de uso
 - 1.1Lenguajes soportados
 - 1.2Parámetros de configuración
 - 1.3Definición de estilos
 - 1.4Inclusión de ficheros automática
 - 1.5Problemas de codificación (¡¡¡¡¡IMPORTANTE EN ESPAÑOL!!!)
 - 1.6Personalización de los títulos

El paquete *listings*, modo de uso[\[editar\]](#)

El paquete `listings` es el de uso habitual para la adición de código fuente de múltiples lenguajes de programación en un documento. Su funcionamiento es parecido al paquete `verbatim`.

Para utilizar este paquete, se debe utilizar:

```
\usepackage{listings}
```


Este paquete permite el resaltado de los lenguajes de programación más comunes, además de ser configurable. Lo único que se necesita es utilizar el entorno `lstlisting`:

```
\begin{lstlisting}
El código se inserta aquí.
\end{lstlisting}
```

También es posible, y es muy útil, insertar desde un fichero de código fuente. Esto facilita al máximo las modificaciones y pruebas del código a insertar mientras aún lo estás editando ya que las modificaciones que se hagan en el código se reflejarán de forma automática en el documento LaTeX (principio DRY):



Para más información, véase el artículo
«[DRY](#)» en [Wikipedia](#).

```
\lstinputlisting{nombre_fichero_codigo.py}
```

En este ejemplo se insertaría un fichero de código en Python. Lo relevante es que se puede incluir cualquier fichero indicando su nombre completo. El lenguaje que se reconoce no depende de la extensión del fichero ni de ninguna indicación interna al mismo, sino que depende de los ajustes que haya establecido para este paquete en el documento. Por ejemplo, para especificar que se trata del lenguaje Python es necesario expresar lo siguiente:

```
\lstinputlisting[language=Python]{nombre_fichero_codigo.py}
```

También es posible seleccionar qué líneas se desean incluir:

```
\lstinputlisting[language=Python, firstline=37,
lastline=45]{nombre_fichero_codigo.py}
```

Lo que es muy útil para mostrar en el documento solamente aquello que sea relevante del código fuente. Si se omiten alguno de estos parámetros, `firstline` o `lastline`, que se deben mostrar todas las líneas desde el comienzo o hasta el final, respectivamente.

El siguiente es un documento completo que inserta un programa en Pascal:

```

\documentclass{article}
\usepackage{listings}           % Incluye el paquete listings
\usepackage[spanish]{babel}
\usepackage[T1]{fontenc}

\begin{document}
\lstset{language=Pascal}       % Establece el lenguaje por defecto. Se
                               % puede cambiar para cada bloque de código insertado

\begin{lstlisting}[frame=single] % Inicia el bloque de código
for i:=maxint to 0 do
begin
{ no hace nada }
end;
Write("Insensible a las mayúsculas");
Write("en las palabras reservadas de Pascal.");
\end{lstlisting}

\end{document}

```

```

begin
{ no hace nada }
end;
Write("Insensible a las mayúsculas");
Write("en las palabras reservadas de Pascal.");

```

Lenguajes soportados[\[editar\]](#)

El paquete `listings` es capaz de resaltar los siguientes lenguajes:

ABAP^{2,4}, ACSL, Ada⁴, Algol⁴, Ant, Assembler^{2,4}, Awk⁴, bash, Basic^{2,4}, C#⁵, C++⁴, C⁴, Caml⁴, Clean, Cobol⁴, Comal, csh, Delphi, Eiffel, Elan, erlang, Euphoria, Fortran⁴, GCL, Gnuplot, Haskell, HTML, IDL⁴, inform, Java⁴, JVMIS, ksh, Lisp⁴, Logo, Lua², make⁴, Mathematica^{1,4}, Matlab, Mercury, MetaPost, Miranda, Mizar, ML, Modelica³, Modula-2, MuPAD, NASTRAN, Oberon-2, Objective C⁵, OCL⁴, Octave, Oz, Pascal⁴, Perl, PHP, PL/I, Plasm, POV, Prolog, Promela, Python, R, Reduce, Rexx, RSL, Ruby, S⁴, SAS, Scilab, sh, SHELXL, Simula⁴, SQL, tcl⁴, TeX⁴, VBScript, Verilog, VHDL⁴, VRML⁴, XML, XSLT.

Para algunos de ellos, se permiten diferentes dialectos. Para más información, utiliza la documentación que viene con el paquete, debería estar en tu distribución, bajo el nombre `listings-*.dvi`.

Notas

1. Soporta Mathematica solamente si se trata de un fichero en texto plano. No se pueden incluir ficheros *.NB., sin embargo, Mathematica puede exportar ficheros fuente formateados para su uso en LaTeX.
2. En estos lenguajes es obligatorio especificar el dialecto del lenguaje (por ejemplo: `language={ [x86masm]Assembler }`).
3. Modelica se soporta mediante el paquete [dtsyntax](#) disponible [aquí](#).
4. Para estos lenguajes hay múltiples dialectos soportados: C, por ejemplo, puede ser ANSI, Handel, Objective y Sharp. Ver p. 12 del [manual de listings](#) para una introducción a esto.
5. Definido como un dialecto de otro lenguaje.

Parámetros de configuración[\[editar\]](#)

Se pueden modificar los parámetros que afectan a cómo se muestra el código. Estos parámetros de configuración se pueden insertar en cualquier parte del documento, antes o después de `\begin{document}`). A continuación se muestra un ejemplo con todos los parámetros que se puede utilizar como base para modificarlo según las necesidades. El significado de cada parámetro se explica en la propia línea como un comentario.

```
\documentclass[a4paper,12pt]{article}
\usepackage[spanish]{babel}
\usepackage[T1]{fontenc}

\usepackage{listings}
\usepackage{color}

\definecolor{miverde}{rgb}{0,0.6,0}
\definecolor{migris}{rgb}{0.5,0.5,0.5}
\definecolor{mimalva}{rgb}{0.58,0,0.82}

\lstset{ %
  backgroundcolor=\color{white},    % Indica el color de fondo; necesita
  que se añada \usepackage{color} o \usepackage{xcolor}
  basicstyle=\footnotesize,        % Fija el tamaño del tipo de letra
  utilizado para el código
  breakatwhitespace=false,         % Activarlo para que los saltos
  automáticos solo se apliquen en los espacios en blanco
  breaklines=true,                 % Activa el salto de línea automático
  captionpos=b,                    % Establece la posición de la leyenda
  del cuadro de código
  commentstyle=\color{miverde},    % Estilo de los comentarios
```

```

deletekeywords={...},           % Si se quiere eliminar palabras clave
del lenguaje
escapeinside={\%*}{*}),        % Si quieres incorporar LaTeX dentro
del propio código
extendedchars=true,            % Permite utilizar caracteres
extendidos no-ASCII; solo funciona para codificaciones de 8-bits; para
UTF-8 no funciona. En xelatex necesita estar a true para que funcione.
frame=single,                  % Añade un marco al código
keepspace=true,                % Mantiene los espacios en el texto.
Es útil para mantener la indentación del código (puede necesitar
columns=flexible).
keywordstyle=\color{blue},      % estilo de las palabras clave
language=Pascal,               % El lenguaje del código
otherkeywords={*,...},         % Si se quieren añadir otras palabras
clave al lenguaje
numbers=left,                  % Posición de los números de línea
(none, left, right).
numbersep=5pt,                 % Distancia de los números de línea al
código
numberstyle=\small\color{migris}, % Estilo para los números de línea
rulecolor=\color{black},        % Si no se activa, el color del marco
puede cambiar en los saltos de línea entre textos que sea de otro color,
por ejemplo, los comentarios, que están en verde en este ejemplo
showspaces=true,               % Si se activa, muestra los espacios
con guiones bajos; sustituye a 'showstringspaces'
showstringspaces=false,        % subraya solamente los espacios que
estén en una cadena de esto
showtabs=true,                 % muestra las tabulaciones que existan
en cadenas de texto con guión bajo
stepnumber=2,                  % Muestra solamente los números de
línea que corresponden a cada salto. En este caso: 1,3,5,...
stringstyle=\color{mimalva},    % Estilo de las cadenas de texto
tabsize=2,                     % Establece el salto de las
tabulaciones a 2 espacios
title=\lstname                  % muestra el nombre de los ficheros
incluidos al utilizar \lstinputlisting; también se puede utilizar en el
parámetro caption
}
\begin{document}

```

```

\begin{lstlisting}[frame=single] % Inicia el bloque de código
  for i:=maxint to 0 do
  begin
    {      comentario
      %* \textbf{texto en negrita} *)
    no hace nada }
  end;
  Write('Insensible a las mayúsculas');
  Write('en las palabras reservadas de Pascal.');
```

```

\end{lstlisting}
\end{document}

```

escapeinside

El parámetro `escapeinside` necesita explicación. La opción `escapeinside={A}{B}` define los delimitadores de inicio y fin que se utilizarán para permitir la introducción de código LaTeX entre el código, en este caso, que todo el código entre "A" y "B" será interpretado como código LaTeX utilizando el estilo vigente en el entorno *listings*. En este ejemplo, se interpretan como código LaTeX a partir de `%*`, hasta que se encuentre el cierre con `*)`.

Hay muchas más opciones, se recomienda revisar la documentación oficial.

Definición de estilos[\[editar\]](#)

Este paquete permite definir estilos. Es decir, perfiles que definen un conjunto de parámetros.

Por ejemplo:

```

\lstdefinestyle{customc}{
  belowcaptionskip=1\baselineskip,
  breaklines=true,
  frame=L,
  xleftmargin=\parindent,
  language=C,
  showstringspaces=false,
  basicstyle=\footnotesize\ttfamily,
  keywordstyle=\bfseries\color{green!40!black},
  commentstyle=\itshape\color{purple!40!black},
  identifierstyle=\color{blue},
  stringstyle=\color{orange},
}

\lstdefinestyle{customasm}{
  belowcaptionskip=1\baselineskip,
  frame=L,

```

```

xleftmargin=\parindent,
language=[x86masm]Assembler,
basicstyle=\footnotesize\ttfamily,
commentstyle=\itshape\color{purple!40!black},
}

\lstset{escapechar=@,style=customc}

```

En este ejemplo, se definen dos estilos y se establece uno de ellos 'customc' como el estilo por defecto y la @ como el carácter de escape.

Se puede utilizar así:

```

\begin{lstlisting}
#include <stdio.h>
#define N 10
/* Comentario en un
 * bloque*/

int main()
{
    int i;

    // Line comment.
    puts(";Hola Mundo!");

    for (i = 0; i < N; i++)
    {
        puts(";LaTeX también es genial para los programadores!");
    }

    return 0;
}
\end{lstlisting}

\lstinputlisting[caption=Scheduler, style=customc]{hello.c}

```

El trozo de código C se imprimirá como:

```

#include <stdio.h>
#define N 10
/* Comentario en un
   * bloque*/

int main()
{
    int i;

    // Line comment.
    puts("¡Hola Mundo!");

    for (i = 0; i < N; i++)
    {
        puts("¡LaTeX también es genial para los
    }

    return 0;
}

```

El código se ha compilado con XeLaTeX para facilitar que las exclamaciones funcionen en el trozo de código C.

Inclusión de ficheros automática[\[editar\]](#)

Si se tiene un conjunto de ficheros de código fuente para incluir, se puede uno encontrar repitiendo una y otra vez el mismo texto LaTeX Aquí es donde las macros muestran toda su potencia:

```

\newcommand{\includecode}[2][c]{\lstinputlisting[caption=#2, escapechar=,
style=custom#1]{#2}<!-->}
% ...

\includecode{sched.c}
\includecode[asm]{sched.s}

```

```
% ...
```

```
\lstlistoflistings
```

Con este ejemplo, se crea un comando para facilitar al máximo la inclusión de código. Se establece un estilo por defecto para que sea *customc*. Todos los listados tendrán su nombre el el título: no hay que escribir el nombre del fichero dos veces gracias a la macro.

Problemas de codificación (¡¡¡¡¡IMPORTANTE EN ESPAÑOL!!!)[[editar](#)]

En el ejemplo anterior, para evitar el problema de codificación que surge con LaTeX y el uso de utf8 dentro de `listings`, se ha compilado los ejemplos con XeLaTeX.

El problema es que, por defecto, `listings` no soporta codificación multibyte para el código fuente.

Para poder manejar UTF-8, resulta necesario decirle a `listings` cómo interpretar los caracteres especiales que surjan entre el código fuente, mediante la definición de los mismos.

Esto se puede hacer de la siguiente forma:

```
\lstset{iterate=
  {á}{\`a}}1 {é}{\`e}}1 {í}{\`i}}1 {ó}{\`o}}1 {ú}{\`u}}1
  {Á}{\`A}}1 {É}{\`E}}1 {Í}{\`I}}1 {Ó}{\`O}}1 {Ú}{\`U}}1
  {à}{\`a}}1 {è}{\`e}}1 {ì}{\`i}}1 {ò}{\`o}}1 {ù}{\`u}}1
  {À}{\`A}}1 {È}{\`E}}1 {Ì}{\`I}}1 {Ò}{\`O}}1 {Ù}{\`U}}1
  {ä}{\`a}}1 {ë}{\`e}}1 {ï}{\`i}}1 {ö}{\`o}}1 {ü}{\`u}}1
  {Ä}{\`A}}1 {Ë}{\`E}}1 {Ï}{\`I}}1 {Ö}{\`O}}1 {Ü}{\`U}}1
  {â}{\`a}}1 {ê}{\`e}}1 {î}{\`i}}1 {ô}{\`o}}1 {û}{\`u}}1
  {Â}{\`A}}1 {Ê}{\`E}}1 {Î}{\`I}}1 {Ô}{\`O}}1 {Û}{\`U}}1
  {œ}{\`oe}}1 {Œ}{\`OE}}1 {æ}{\`ae}}1 {Æ}{\`AE}}1 {ß}{\`ss}}1
  {ű}{\`H{u}}1 {Ű}{\`H{U}}1 {ő}{\`H{o}}1 {Ő}{\`H{O}}1
  {ç}{\`c c}}1 {Ç}{\`c C}}1 {ø}{\`o}}1 {å}{\`r a}}1 {Å}{\`r A}}1
  {€}{\`EUR}}1 {£}{\`pounds}}1
}
```

La tabla anterior cubre la mayoría de los caracteres de los idiomas latinos:

Para una explicación más detallada del uso de `iterate` se puede revisar la sección 6.4 en la [Documentación de listings](#).

Otra posibilidad es sustituir el paquete `\usepackage{listings}` (en el preámbulo) por `\usepackage{listingsutf8}`.

Personalización de los títulos[[editar](#)]

Se pueden tener unos títulos (caption y/o title) muy avanzados gracias al paquete `caption`. Veamos un ejemplo de uso con `listings`.


```

\usepackage{caption}
\usepackage{listings}

\DeclareCaptionFont{white}{ \color{white} }
\DeclareCaptionFormat{listing}{
  \colorbox[cmymk]{0.43, 0.35, 0.35,0.01 }{
    \parbox{\textwidth}{\hspace{15pt}\#1\#2\#3}
  }
}
\captionsetup[lstlisting]{ format=listing, labelfont=white,
textfont=white, singlelinecheck=false, margin=0pt, font={bf,footnotesize}
}

% ...

\lstinputlisting[caption=My caption]{sourcefile.lang}

```

Manual de LaTeX/Escribiendo texto/Alineación del texto

< [Manual de LaTeX](#) | [Escribiendo texto](#)

automáticamente justificado los párrafos (es decir, los alinea por ambos lados), aunque podemos alinear el texto solo por la izquierda con la declaración `\raggedright` (bandera por la derecha) y solo por la derecha con la declaración `\raggedleft` (bandera por la izquierda). Para centrar el texto hemos de usar la declaración `\centering`.

Alternativamente podemos usar los entornos de alineación `flushleft`, `flushright` y `center`. Puesto que estos son entornos, para, por ejemplo, centrar texto con `center` debemos escribir

```
\begin{center}
```

```
Texto centrado
```

```
\end{center}
```

Manual de LaTeX/La estructura de un documento en LaTeX/Preámbulo/Clases de documento

< [Manual de LaTeX](#) | [La estructura de un documento en LaTeX](#) | [Preámbulo](#)

[La estructura de un documento en LaTeX/Preámbulo](#)

[← TEMA ANTERIOR](#)

Clases de documento

[ÍNDICE](#)

Como mencionábamos, existen diferentes estilos que podemos darle a nuestro documento, y la selección de uno u otro dependerá de qué es lo que necesitamos hacer. Si queremos escribir un documento corto, podemos utilizar la clase `article`. En términos generales, esta clase de documento nos permite dividir el documento en secciones, subsecciones, párrafos y subpárrafos.

A continuación una lista de algunas clases típicas de documento:

- article** Para artículos académicos y otros documentos cortos que no es necesario dividir en capítulos, sino que bastan las secciones y subsecciones y sus párrafos y subpárrafos.
- book** Para libros y otros documentos más largos que deben incluir capítulos, prólogo, apéndices o incluso partes.
- report** Para informes técnicos. Es similar a la clase `book`.
- memoir** Una clase todoterreno con un buen número de funciones adicionales integradas.
- beamer** Otra clase para presentaciones mediante diapositivas.

Las clases `book` y `report` son muy similares, y ambas sirven para documentos grandes, como lo son, naturalmente, los libros y los reportes, entre otros trabajos. Sin embargo, existen ligeras diferencias. Por ejemplo, la clase `book` hace que los capítulos empiecen siempre en una página impar, de modo que si un capítulo anterior termina en una página impar, la página (par) siguiente quedará en blanco y el capítulo nuevo comenzará después de ella. Esto, en cambio, no sucede con la clase `report`, así es que un capítulo simplemente empieza en una página nueva, sea par o impar. Por supuesto, estas opciones pueden ser fácilmente modificadas. Todas las clases de la lista anterior admiten opciones adicionales. Por ello, la sintaxis general para indicar una clase de documento es la siguiente:

```
\documentclass['opción 1, opción 2, ...']['clase de documento']
```

Las opciones que podemos dar son:

a4paper, letterpaper, ...	<p>Con esta opción indicamos que el tamaño del papel debe de ser <code>a4paper</code> (tamaño a4), <code>letterpaper</code> (tamaño carta), ... Otras opciones que determinan distintos tamaños de página son:</p> <ul style="list-style-type: none"> • <code>a5paper</code> (210 mm 148 mm) • <code>b5paper</code> (250 mm 176 mm) • <code>legalpaper</code> (14 in 8.5 in) • <code>executivepaper</code> (10.5 in 7.25 in) <p>El valor por defecto es <code>letterpaper</code>, de Estados Unidos y México. En los documentos de otros países puede ser necesaria la opción <code>a4paper</code>.</p>
landscape	Apaisado. Pone la página de forma horizontal.
10pt, 11pt, 12pt	Definen el tamaño de la fuente principal del texto.
oneside, twoside	Indican si el documento debe estar adaptado a impresión por un sólo lado de la página o por ambos lados de ella.
titlepage, notitlepage	Determinan si el documento debe o no incluir una página de título, i.e. si va a incluir o no una portada.
openright, openany	<code>openright</code> obliga a los capítulos a iniciar siempre sólo en páginas impares, mientras que con la opción <code>openany</code> permitimos que los capítulos se inicien en cualquier página.
onecolumn, twocolumn	Definen si el documento se va a escribir en una sola columna o a doble columna.
fleqn	Esta opción hace que las ecuaciones queden alineadas por la izquierda en lugar de que sean centradas (como sucede por defecto).
leqno	Con esta opción hacemos que el número de las ecuaciones quede alineado por la izquierda en lugar de por la derecha (como sucede por defecto).
draft, final	La opción <code>draft</code> se usa si queremos que la compilación del documento se haga a modo de "borrador". Con <code>draft</code> haremos que las líneas que sean demasiado largas queden marcadas mediante cajas negras. La opción <code>final</code> producirá simplemente que el documento se compile de manera normal.

Cuando no especificamos opciones para una clase de documento, se cargan las opciones por defecto de la clase que estemos utilizando. Por ejemplo, si escribimos

```
\documentclass[letterpaper,10pt,twoside,onecolumn,final,openright]{book}
```

sería lo mismo que si escribiéramos simplemente

```
\documentclass{book}
```

pues la clase `book` tiene como opciones por defecto `letterpaper,10pt,twoside,onecolumn,final,openright`. Además, la clase `book` producirá automáticamente una página para el título del documento. Con la opción `notitlepage` haremos que esto no suceda así, de manera que el título del documento no quedará en una página aparte.

La clase `article` carga automáticamente las opciones `letterpaper,10pt,oneside,onecolumn,final`. Puesto que en la clase `article` no existen capítulos, las opciones `openright` y `openany` no están permitidas.

Las opciones por defecto de la clase `report` son `letterpaper,10pt,oneside,final,openany`.

Diseño del documento[\[editar\]](#)

Tanto los márgenes como el tamaño del papel se pueden cambiar a los valores que se deseen con el paquete `geometry`. Un ejemplo simple, que ajusta todos los márgenes a 1 cm en una hoja DIN A5, es:

```
\usepackage[a5paper,margin=1cm]{geometry}
```

Un paquete alternativo es `zwpagelayout`, con menos opciones, pero que ajusta internamente los parámetros necesarios en un PDF (y que tiene otras funciones como marcas de corte, por ejemplo).

Manual de LaTeX/Escribiendo texto/Tamaños, estilos y tipos de letra

< [Manual de LaTeX](#) | [Escribiendo texto](#)

En las letras (o fuentes) tienen en general 5 atributos, aunque sólo mencionaremos cuatro de ellos, que son los que determinan el aspecto del carácter en el texto compilado.

Para los cambios breves de la fuente, lo recomendable es:

```
Este es un texto que puede tener \textit{cursiva} y también
\textbf{negrita}.
También puede ser \textit{\textbf{cursiva con negrita}}. Otra
posibilidad
es la \textsc{versalita}.
```

La conversión de caja se obtiene con `\MakeUppercase{texto}` (que convierte a mayúscula) y `\MakeLowercase{TEXTO}` (a minúscula). (No deben usarse en LaTeX las órdenes `\uppercase` y `\lowercase`, aunque las admita, porque no siempre dan el resultado correcto.)

Sumario

[ocultar]

- 1Familia
- 2Serie
- 3Forma
- 4Tamaño

Familia[\[editar\]](#)

La **familia** es el nombre de una colección de fuentes. organiza las fuentes en tres familias, que son [Archivo:Roman.svg](#), [Archivo:Sans Serif.svg](#), y



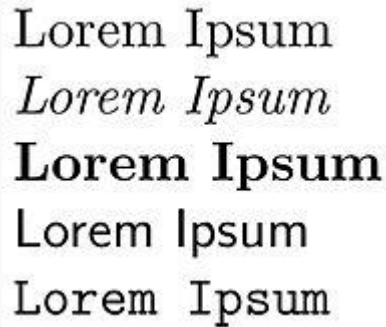
. Para conseguir cada una de estas familias se usan, respectivamente, los comandos `\rmfamily` (letras con remates), `\sffamily` (letras sin remates) y `\ttfamily` (letras mecanográficas). Estos comandos son en realidad *declaraciones*, por lo que su efecto se limita de manera distinta a la de los comandos comunes. Por ejemplo, si queremos conseguir un texto con caracteres



, debemos escribir

```
{\ttfamily ''texto''}
```

y así el efecto de `\ttfamily` afectará sólo al texto que se encuentre entre llaves.



Muestras de la familia Computer Modern

Las fuentes preferidas de LaTeX pertenecen a la familia Computer Modern, pero podrían cambiarse a otras como, respectivamente, Times, Helveticas y Courier, por ejemplo. Para ello lo recomendado es cargar algún paquete, pero para emplear una fuente arbitraria instalada en el sistema es necesario recurrir a dos variantes de TeX llamadas XeTeX y LuaTeX. X

Serie[[editar](#)]

La **serie** de una fuente determina que tan gruesa o expandida será ésta.

Con `\mdseries` tenemos la opción Medium (media) y la opción **Bold**(negrita). Caracteres con este tipo de series se consiguen, respectivamente, con las declaraciones `\mdseries` y `\bfseries`. Como éstas también son declaraciones, para obtener, por ejemplo, un texto en negritas hemos de escribir `{\bfseries ''texto''}`.

Forma[[editar](#)]

La **forma** que puede tener un carácter dentro de una familia puede ser: [Archivo:Upright.svg](#) (vertical o recta), [Archivo:Italic.svg](#) (itálica), [Archivo:Slanted.svg](#) (inclinada) o [Archivo:Small Caps.svg](#) (Mayúsculas y mayúsculas pequeñas). Estas formas se consiguen con las declaraciones `\upshape`, `\itshape`, `\slshape` y `\scshape`, respectivamente.

Además, tenemos los comandos

```
\textbf{''t  
exto''}:
```

 para texto en negritas

```
\textit{''t  
exto''}:
```

 para texto en itálicas

```
\textsl{'t  
exto'}:
```

para texto inclinado

```
\texttt{'t  
exto'}:
```

para texto en
estilo



`\textsc{'t`
exto''}:

para texto en mayúsculas y minúsculas pequeñas

Nota. Aunque LaTeX no dé error con las órdenes de Plain TeX `\bf`, `\it`, `\sf`, etc., no deberían usarse en lugar de las recién descritas.

Tamaño[[editar](#)]

El tamaño de una letra puede ser

<code>{\tiny tiny}</code>	que se consigue con la declaración <code>\tiny</code>
<code>{\scriptsize scriptsize}</code>	que se consigue con la declaración <code>\scriptsize</code>
<code>{\small small}</code>	que se consigue con la declaración <code>\small</code>
	que se consigue con la declaración <code>\normalsize</code>
	que se consigue con la declaración <code>\large</code>
<code>{\Large larger}</code>	que se consigue con la declaración <code>\Large</code>
<code>{\LARGE LARGE}</code>	que se consigue con la declaración <code>\LARGE</code>
<code>{\huge huge}</code>	que se consigue con la declaración <code>\huge</code>
<code>{\Huge Huge}</code>	que se consigue con la declaración <code>\Huge</code>

Estas órdenes no solo ajustan el tamaño de la letra, sino también la interlínea y en ocasiones también otros parámetros relacionados con listas y ecuaciones. Un error habitual es escribir un párrafo del siguiente modo:

```
{\small
```

```
Texto texto texto
texto texto.}
```

Aunque con ello se cambia el tamaño de la letra, la interlínea sigue igual. Es necesario señalar un párrafo con, por ejemplo:

```
{\small
  Texto texto texto
  texto texto.\par}
```

Manual de LaTeX/Inclusión de gráficos/Gráficos con TikZ

< [Manual de LaTeX](#) | [Inclusión de gráficos](#)

Sumario

[ocultar]

- [1Introducción](#)
- [2Estructura capas](#)
- [3Dibujo básico](#)
- [4Estilos](#)
- [5Nodos](#)
- [6Otras librerías](#)
 - [6.1Árboles](#)
 - [6.2Gráficos](#)
 - [6.3Calendarios](#)
- [7Referencias](#)

Introducción[\[editar\]](#)

TikZ es un complejo lenguaje gráfico que permite *dibujar* en un documento LaTeX sin ninguna otra herramienta auxiliar.

IMPORTANTE: TikZ tiene un conflicto con el paquete `babel` cuando se trabaja en español, ya que ambos definen como caracteres activos "<" y ">": `babel` para el entrecorillado "multinivel", y `tikz` para las flechas que aparecen al final de una línea. Por ello, es necesario eliminar este conflicto, usando, por ejemplo:

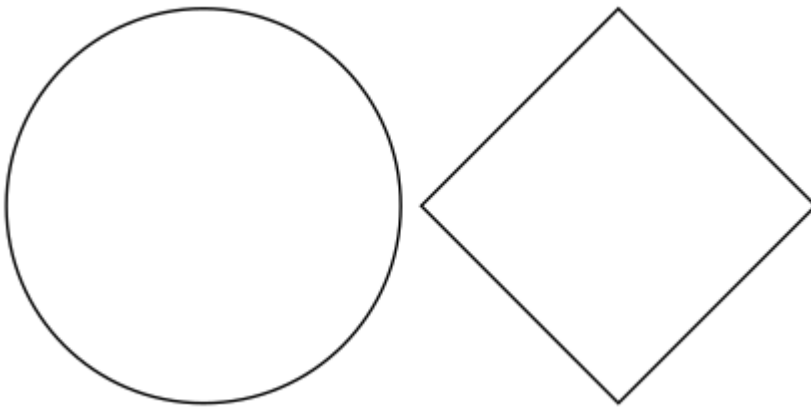
```
\usepackage[spanish,es-noshorthands]{babel}
```

Que suprime el uso "activo" de los caracteres "<" y ">" en babel.

TikZ define un conjunto de comandos y entornos LaTeX elaborar gráficos. Por ejemplo, el código:

```
\tikz \draw (0,0) circle (1cm);  
\tikz \draw (-1,0) -- (0,1) -- (1,0) -- (0,-1) -- cycle;
```

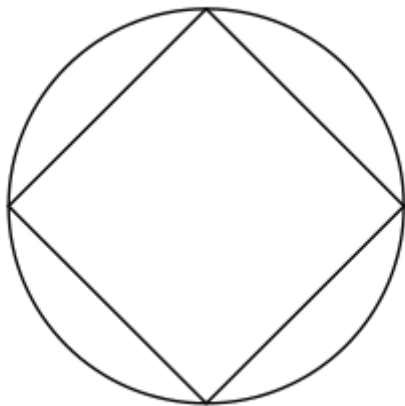
Genera el siguiente dibujo:



Las unidades de TikZ, por defecto, son en centímetros. Por lo que en un caso se dibuja un círculo de un centímetro de diámetro centrado en el punto (0,0), y en el otro se genera un cuadrado de un centímetro de lado. Se observa que el comando `\tikz` genera dibujos independientes entre sí. Para que ambos dibujos sean uno solo, generando una figura con el cuadrado y el círculo superpuestos, en lugar de usar el comando `\tikz`, es necesario incluir los comandos de dibujo en un entorno `\begin{tikzpicture}`, por ejemplo:

```
\begin{tikzpicture}  
\draw (0,0) circle (1cm);  
\draw (-1,0) -- (0,1) -- (1,0) -- (0,-1) -- cycle;  
\end{tikzpicture}
```

Genera lo siguiente:



Las ventajas de usar `\tikz` en lugar de una herramienta externa son, entre otras:

1. Control completo sobre el diagrama.
2. Se usa el mismo tipo de letra que en el documento.
3. Capacidad de utilizar macros.
4. La alta calidad del resultado.

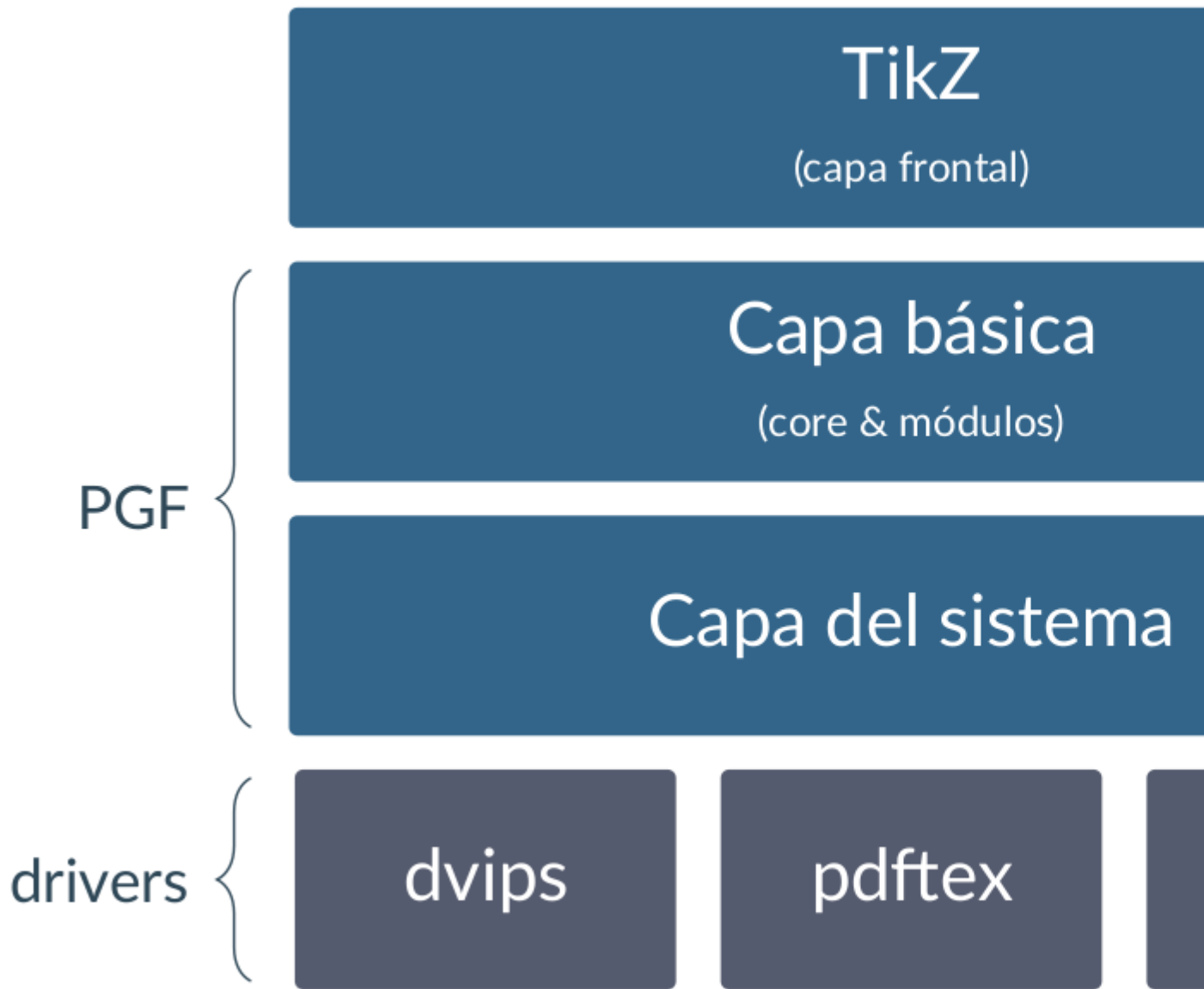
Como inconvenientes, cabe citar:

1. Elevada curva de aprendizaje.
2. No se “dibuja” sobre el propio “dibujo”.
3. El código puede ser complejo de entender.

Entre las ventajas que presenta TikZ frente a otros paquetes de LaTeX se encuentra su independencia del “driver” de impresión, sea postscript o no. Para ello, se definió un lenguaje independiente de representación de gráficos, *PGF*, Portable Graphics Format, que posteriormente, cada uno de los “drivers” existentes, convierten al formato destino elegido, sea postscript, pdf, \ldots

Estructura capas[\[editar\]](#)

TikZ es un lenguaje gráfico que dispone de una capa semántica avanzada que permite diseñar diagramas con relativa facilidad manteniendo en el lenguaje la relación entre los elementos del diagrama. Para conseguirlo, se sustenta en dos capas que pueden utilizarse de forma independiente. En la figura siguiente se puede observar la relación entre ellas.



Capa del sistema.

Esta primera capa es la API básica de PGF que abstrae las diferencias entre los diversos “drivers”. Cada uno de los elementos de esta API se tiene que implementar de forma diferente para cada “driver”, por lo tanto, resulta necesario mantenerla lo más simple posible, a fin de simplificar al máximo el desarrollo y mantenimiento de aquellos.

Capa básica.

Añade semántica que facilita el diseño de gráficos. Esta capa se puede extender con paquetes adicionales.

TikZ.

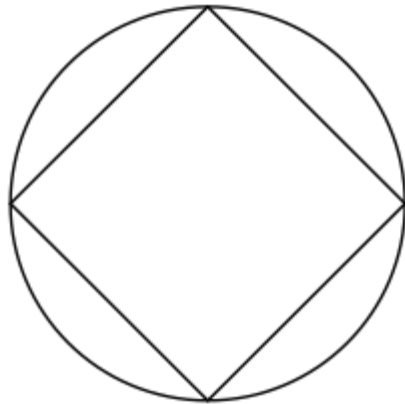
Se trata de una API avanzada, que hace de “frontal” sobre la capa básica simplificando y extendiendo su sintaxis. Su sintaxis se basa en la de METAFONT y PSTricks. Sobre este último presenta la ventaja de ser “portable” entre “drivers”. Lo que, actualmente, se traduce en la generación directa de documentos PDF.

Además de lo visto, PGF dispone de algunos paquetes de utilidades como: `pgfpages`, para “jugar” con las páginas de un documento componiendo varias de ellas en una única física; `pgfkeys`, para la gestión de parejas de parámetros, clave-valor; `pgfcalendar`, para la generación de calendarios con LaTeX, o `pgffor`, que define el comando `foreach`.

Dibujo básico[\[editar\]](#)

Normalmente, un gráfico TikZ se compone utilizando el entorno `tikzpicture`, como se vio en el ejemplo inicial^[1]. Un elemento fundamental en TikZ es el *camino* ^[2]. Existen diversos comandos, como `\draw`, `\filldraw` o `\path` que se basan en la idea de encadenar posiciones y unirlos mediante otros elementos.

En el ejemplo anterior:



```
\begin{tikzpicture}
\draw (0,0) circle (1cm);
\draw (-1,0) -- (0,1) -- (1,0) -- (0,-1) -- cycle;
\end{tikzpicture}
```

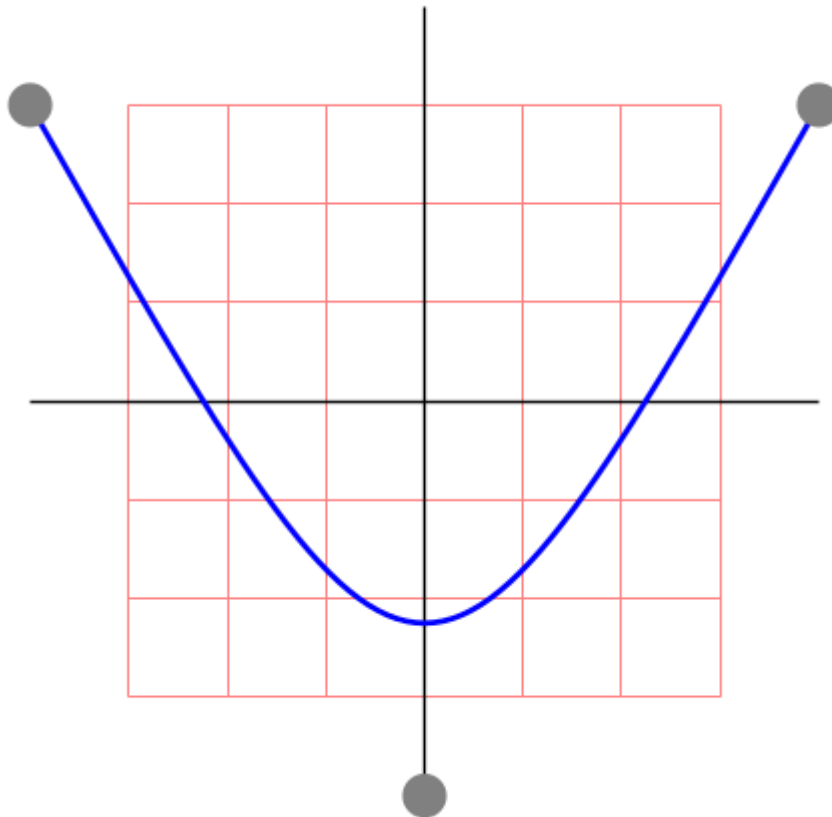
- `\draw` indica que se debe “dibujar” el camino a realizar. Cada comando `\draw` inicia un camino completo nuevo.
- El camino se expresa por una sucesión de puntos y “operaciones” que expresan lo que hay que hacer entre cada punto.

- “(-1,0)” indica que se debe posicionar en dicho punto. En centímetros por defecto.
- Con el comando `draw`, la operación “--” indica que se debe dibujar una línea recta entre el punto anterior y el siguiente.
- Con el comando `draw`, la operación “circle” indica que se debe dibujar un círculo con el radio indicado en el siguiente parámetro, en este caso “(1cm)”.
- “cycle” indica que se vaya al punto de inicio del camino actual.

Existen diversas operaciones para recorrer un camino: `rectangle`, `grid`, `ellipse`, son algunas de las factibles. Cada una de ellas necesita unos parámetros determinados para su correcto dibujo.

Estilos[\[editar\]](#)

Los diferentes comandos tienen parámetros opcionales que facilitan el diseño de diferentes estilos para los gráficos. Por ejemplo:



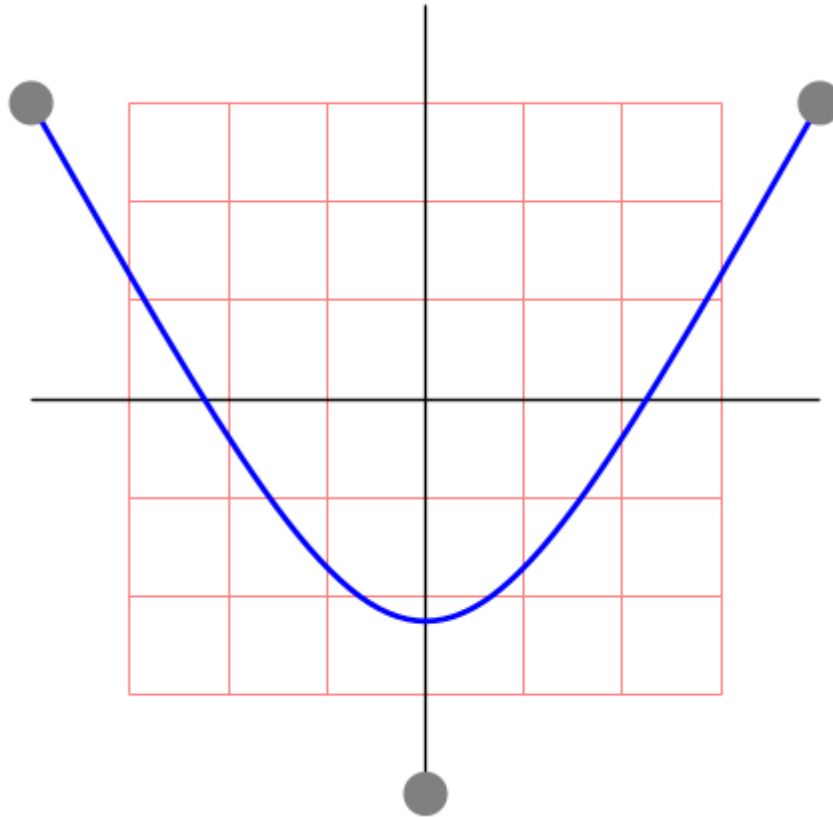
```
\draw[step=0.5,color=red!50!white, very thin] (-1.5,-1.5)
grid (1.5,1.5);
\draw (-2,0) -- (2,0);
\draw (0,-2) -- (0,2);
```

```
\draw[thick, color=blue] (-2,1.5) .. controls (0,-2) .. (2,1.5);
\filldraw[color=gray] (-2,1.5) circle (3pt);
\filldraw[color=gray] (2,1.5) circle (3pt);
\filldraw[color=gray] (0,-2) circle (3pt);
```

Se observa que:

- El primer comando tiene unos parámetros envueltos entre corchetes. Estos parámetros permiten matizar el funcionamiento de las operaciones a realizar en el comando. En concreto, están indicando:
 - `step`, que al dibujar el `grid` la malla tenga un grosor de medio centímetro.
 - `color`, expresa, en formato `xcolor`, que debe dibujar en color rojo claro.
 - `very thin`, expresa el grosor de las líneas que se dibujen en este comando, en este caso concreto, la malla a pintar.
- Los dos siguientes comandos `\draw` dibujan los ejes horizontales con el estilo por defecto (líneas negras de grosor `thin`).
- El siguiente comando `\draw` dibuja una línea desde el punto `(-2,1.5)` al punto `(2,1.5)`, pero en lugar de ser recta, la operación `.. controls (0,-2) ..`, establece que se dibuje la línea utilizando este punto de control como elemento de referencia para trazarla curva. Lo que hace `tikz` es “salir” del punto inicial de forma paralela a la recta imaginaria que llegaría al punto de control, para después girar de tal modo que llegue al punto de destino de forma paralela a la recta imaginaria que va del punto de destino al punto de control. La sintaxis de esta operación permite utilizar dos puntos de control diferentes para el origen y el destino, así tanto el ángulo como la distancia a dichos puntos definen la forma de la curva a trazar. En este comando también se ha especificado un estilo entre corchetes, línea gruesa y de color azul.
- Los tres últimos comandos `\filldraw` dibujan con relleno: se posicionan en un punto; después, la operación a ejecutar es `circle`, con un radio de 3pt, que en este caso se rellenan del mismo color que se ha indicado, `gray`, gris.

Con estas herramientas se pueden realizar dibujos muy complejos. Para facilitar al máximo su desarrollo, `tikz` permite, además, darle nombre a los estilos y reutilizarlos tantas veces como se necesite. Por ejemplo, el dibujo anterior se puede reescribir de la siguiente forma:



```
\begin{tikzpicture}[
  migrid/.style={step=0.5,color=red!50!white, very thin},
  micurva/.style={thick, color=blue},
  mipunto/.style={color=gray}]
\draw[migrid] (-1.5,-1.5) grid (1.5,1.5);
\draw (-2,0) -- (2,0);
\draw (0,-2) -- (0,2);
\draw[micurva] (-2,1.5) .. controls (0,-2) .. (2,1.5);
\filldraw[mipunto] (-2,1.5) circle (3pt);
\filldraw[mipunto] (2,1.5) circle (3pt);
\filldraw[mipunto] (0,-2) circle (3pt);
\end{tikzpicture}
```

- `migrid/.style` define un estilo denominado `migrid` que se puede utilizar entre corchetes en los diferentes comandos para aplicar los parámetros que definen el estilo.
- Después del signo `=`, entre llaves, se describen los parámetros que definen el estilo.
- Esto permite la reutilización como se observa en los tres comandos `\filldraw`.

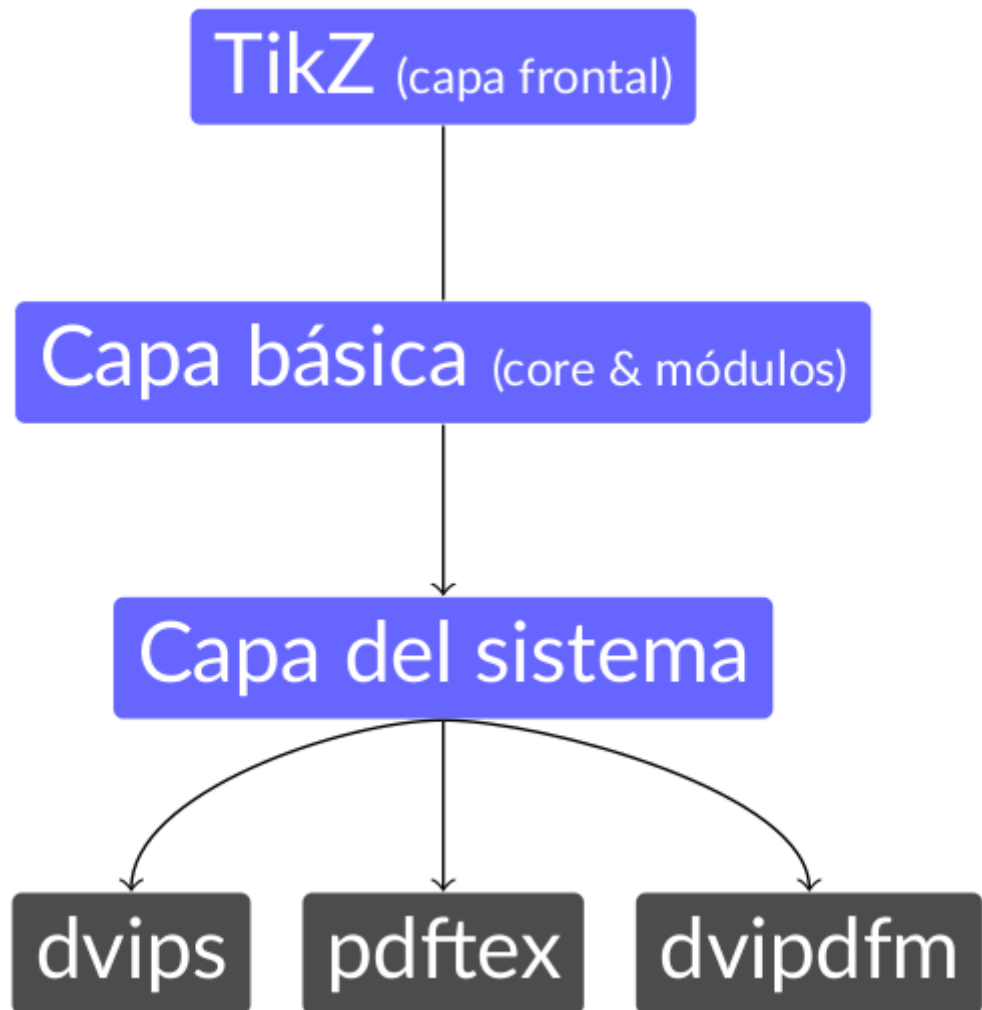
Nodos[\[editar\]](#)

Los nodos facilitan la generación de diagramas que contengan textos o que requieran mostrar relaciones entre elementos, entre otras cosas. A continuación se muestra un ejemplo de uso de los nodos que utiliza algunas librerías adicionales para facilitar su diseño.

- `fit`, para calcular el tamaño de los nodos.
- `positioning`, para colocar los nodos en posiciones relativas a otros.

Para utilizar estas librerías auxiliares de TikZ se utiliza el comando:

```
\usetikzlibrary{fit,positioning}.
```



```
\begin{tikzpicture}[node distance= 1.5cm, auto,
```

```

every node/.style={text=white, rounded corners=0.05cm},
grande/.style={rectangle, fill=blue!60!white, font=\large},
peque/.style={rectangle, fill=white!30!black, font=\large}]

\node[grande](sistema){Capa del sistema};
\node[grande, above of=sistema](basico){Capa básica \scriptsize
(core \& módulos)};
\node[grande, above of=basico](tikz){TikZ \scriptsize (capa
frontal)};
\node[peque, below of=sistema](pdftex){pdftex};
\node[peque, left=0.25cm of pdftex](dvips){dvips};
\node[peque, right=0.25cm of pdftex](dvi pdfm){dvi pdfm};

\draw[->] (tikz.south) -- (basico.north) (basico.south) --
(sistema.north);
\draw[->] (sistema.south) .. controls +(left:0.5cm) and
+(up:0.5cm) .. (dvips.north);
\draw[->] (sistema.south) .. controls +(right:0.5cm) and
+(up:0.5cm) .. (dvi pdfm.north);
\draw[->] (sistema.south) -- (pdftex.north);
\end{tikzpicture}

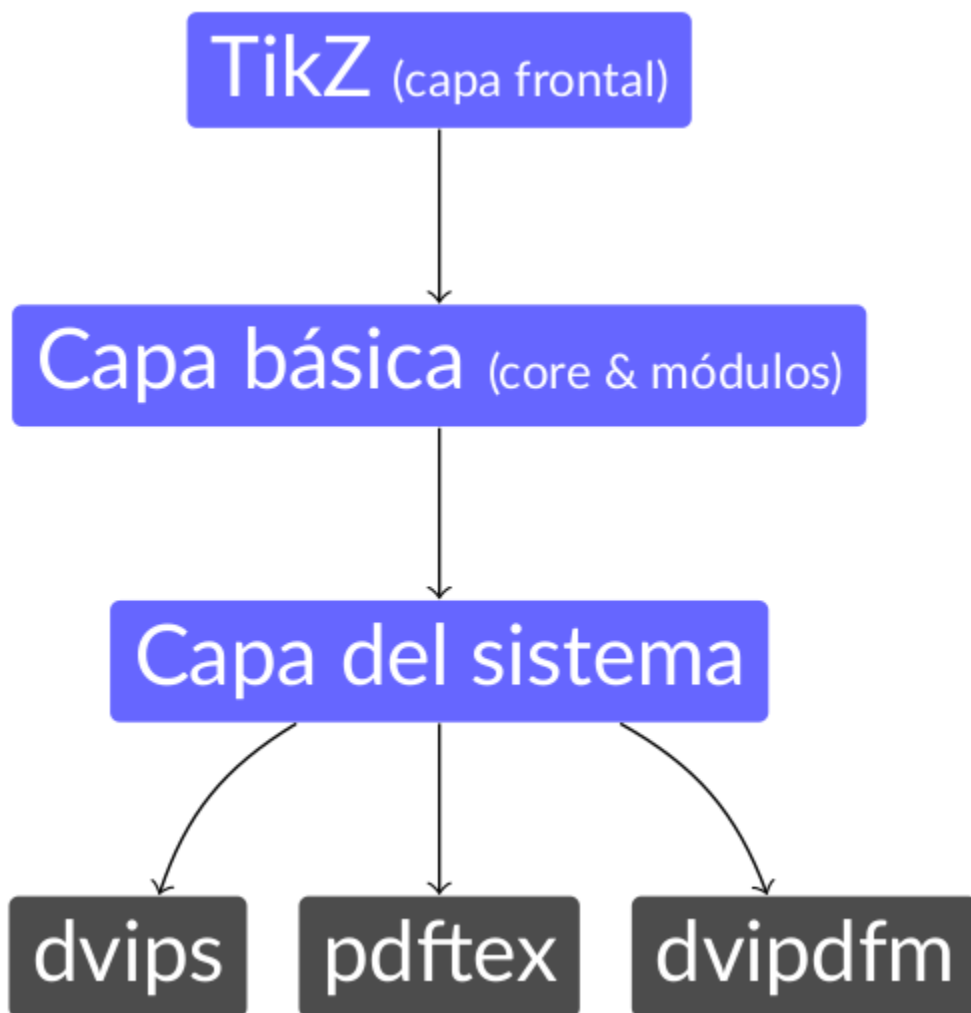
```

- Las primeras líneas establecen los estilos a utilizar: El texto será de color blanco para todos los nodos y las esquinas redondeadas; los nodos con el estilo “grande” utilizan un color de relleno diferente a los pequeños, el tipo de letra será grande y deberán tener forma de “rectangle”.
- En cada comando `\node` se observa, entre paréntesis, el “nombre” que tendrá el nodo, seguido de su contenido entre llaves. El “nombre” se utiliza para referenciar al nodo desde otro lugar. En este caso concreto, para establecer la posición relativa entre ellos.
- Entre los parámetros de cada nodo se incluye el estilo y la posición relativa a otro nodo: “above of”, “below of”, “left” y “right”. Se observa que se puede incluir la distancia relativa de separación entre ellos.
- Para dibujar las líneas que conectan los nodos, se utiliza `draw`. El parámetro especifica el tipo de línea a dibujar, en este caso con una flecha al final (pero no al comienzo) del camino.
- Si se deseara una línea con flecha tanto al comienzo como al final del camino se debería utilizar “<->”. Para expresar una flecha solo al comienzo, se utilizar “<-”.
- Para indicar los lugares de origen y de destino de la línea, se indican los nombres de los respectivos nodos, seguidos de un punto y del modificador que expresa el punto del que debe salir: “north”, desde el borde superior; “south”, desde el inferior; “east”, desde el borde derecho, y “west”, desde el

borde izquierdo. Si no se indican los puntos de inicio y fin de la línea, TikZ busca los más próximos entre sí.

- Para expresar una línea curva se utiliza `controls` como se vio anteriormente. En este caso se indican dos puntos de control. Ambos, precedidos del símbolo “+”, que sirve para expresar coordenadas relativas, a los puntos de salida y llegada respectivamente.
- Cada punto de control, además de ser relativo, indica posición, no mediante un punto (x,y) sino mediante un desplazamiento a través de un eje. Este se expresa con el texto “left” o “right”, en este caso.

TikZ es tan potente que permite expresar el mismo dibujo de diversas maneras. En este caso se puede hacer uso de la operación `edge` que facilita la unión entre nodos, sin necesidad de indicar `draw`. De este modo se captura la semántica entre nodos y enlaces. El código queda así:



```
\begin{tikzpicture}[node distance= 1.5cm, auto,  
  
every node/.style={text=white, rounded corners=0.05cm},  
grande/.style={rectangle, fill=blue!60!white, font=\large},
```



```

peque/.style={rectangle, fill=white!30!black, font=\large}]

\node[grande](sistema){Capa del sistema}
edge[->, bend right=20] (dvips)
edge[->] (pdftex)
edge[->, bend left=20] (dvipdfm);
\node[grande, above of=sistema](basico){Capa básica \scriptsize
(core \& módulos)}
edge[->] (sistema);
\node[grande, above of=basico](tikz){TikZ \scriptsize (capa
frontal)}
edge[->] (basico);
\node[peque, below of=sistema](pdftex){pdftex};
\node[peque, left=0.25cm of pdftex](dvips){dvips};
\node[peque, right=0.25cm of pdftex](dvipdfm){dvipdfm};
\end{tikzpicture}

```

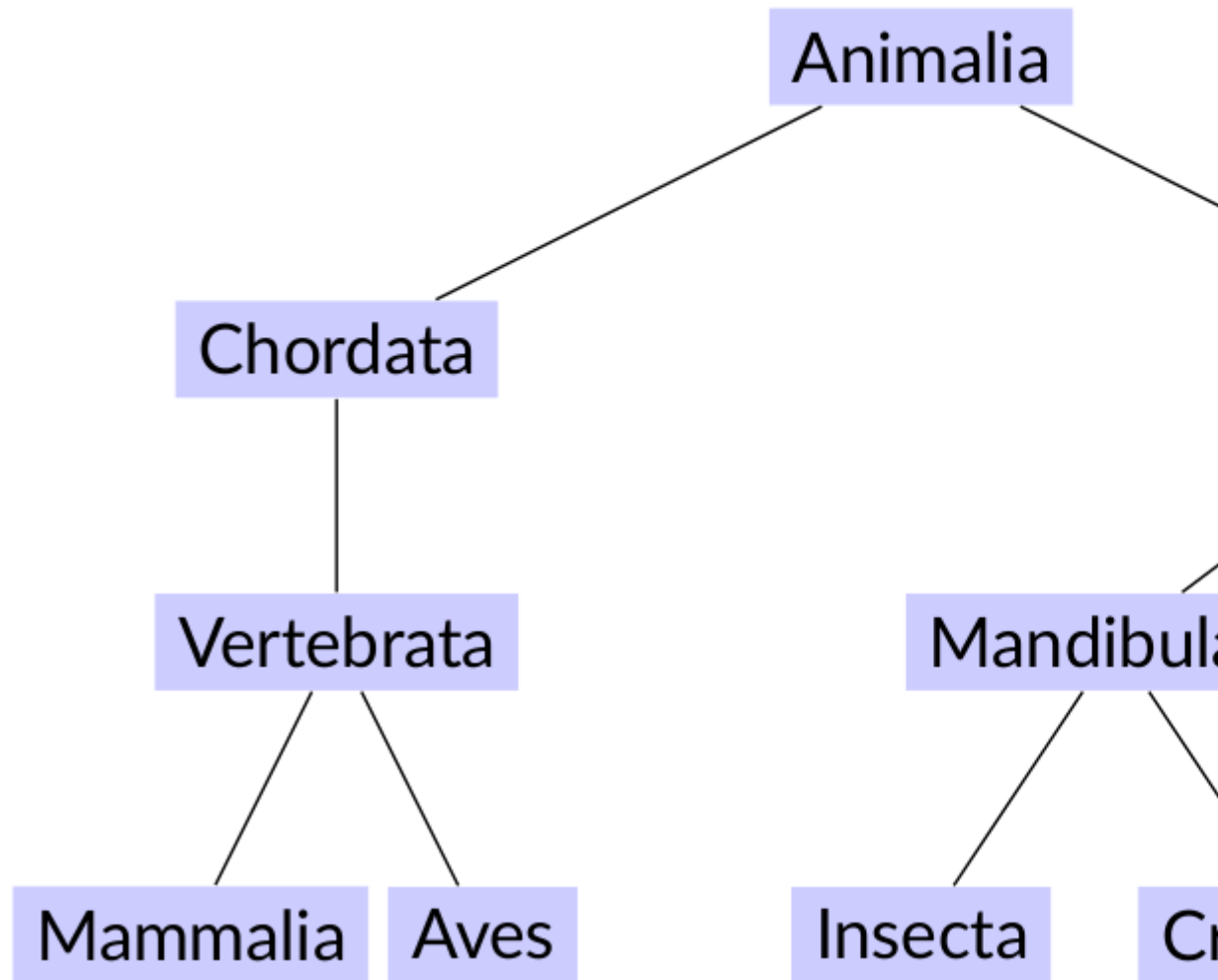
Como se observa, la operación `edge`, en el comando `\node`, facilita el dibujo del enlace entre el nodo que se está definiendo y otro que se indica por su nombre. Además, es posible expresar el “grado de curvatura” del enlace y TikZ se encargará de posicionar la curva en el mejor lugar posible.

Otras librerías[\[editar\]](#)

Como se ve, TikZ es un paquete muy potente. Con lo visto anteriormente se ha mostrado de forma superficial parte de esta potencia. Sin embargo, existen muchos parámetros, operaciones y comandos que no se han mostrado. Además, existen muchas librerías que proporcionan la semántica apropiada para dibujar determinados tipos de diagramas. Por ejemplo, grafos, redes de petri, capas, mapas mentales, calendario, gráficos de funciones, etc. A continuación se muestran algunos ejemplos en los que se observará cómo la semántica de TikZ describe perfectamente la estructura del diagrama correspondiente:

Árboles[\[editar\]](#)

En TikZ se pueden generar árboles como el que sigue:



El código que lo genera es el siguiente:

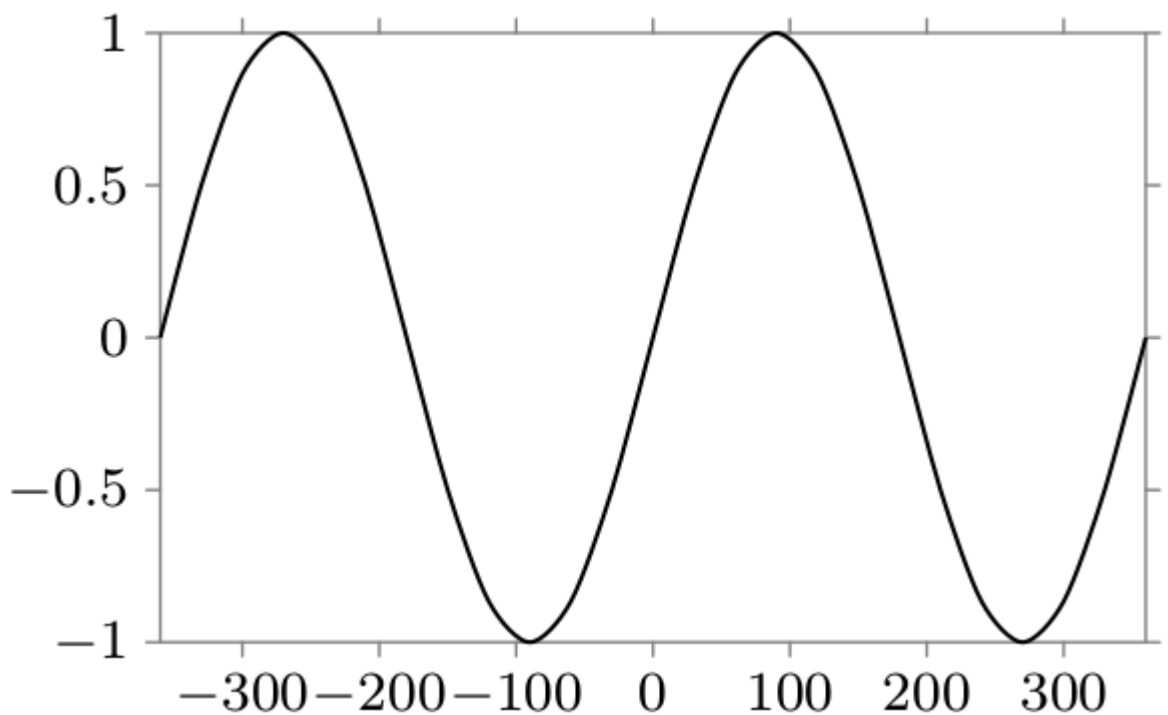
```
\begin{tikzpicture}[every node/.style={rectangle,
fill=blue!20!white}]
\node {Animalia} [sibling distance=6cm]
child {node {Chordata}
child {node {Vertebrata} [sibling distance=1.5cm]
child {node {Mammalia}}
child {node {Aves}}
}
}
child {node {Arthropoda} [sibling distance=4cm]
child {node {Mandibulata}[sibling distance=2cm]
child {node {Insecta}}
child {node {Crustacea}}
}
}
child {node {Chelicerata}}
```

```
child {node {Arachnida}}
}
};
\end{tikzpicture}
```

Los árboles se pueden dibujar también en horizontal. Existe, además, una librería, `mindmap`, de TikZ que facilita la construcción de árboles con un diseño más aproximado para los mapas mentales.

Gráficos[\[editar\]](#)

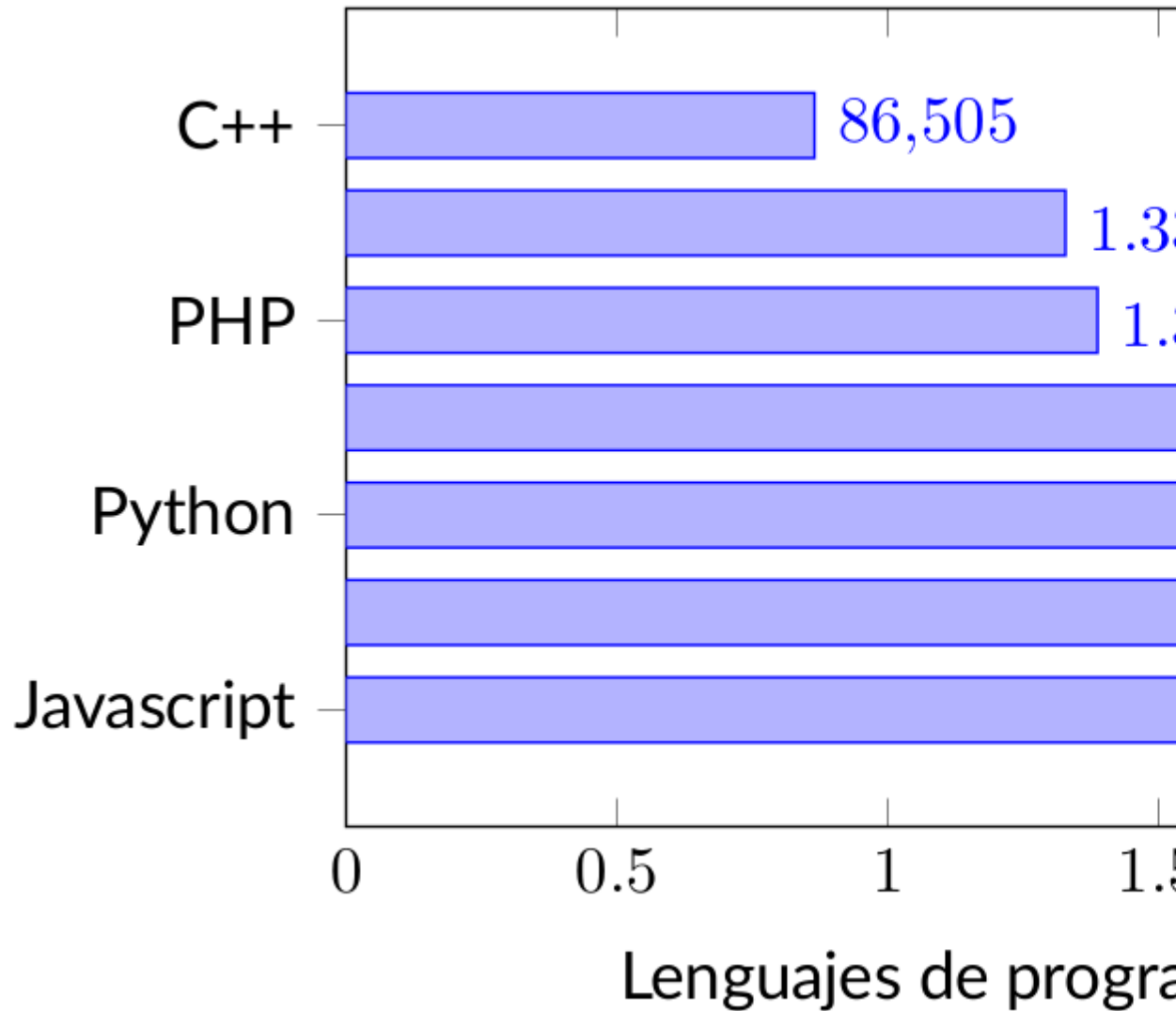
En TikZ se pueden generar gráficos como el que sigue:



El código es el que sigue:

```
\begin{tikzpicture}
\datavisualization [scientific axes, visualize as smooth line]
data [format=function] {
var x : interval [-360:360];
func y = sin(\value x);
};
\end{tikzpicture}
```

Para diagramas más avanzados, como por ejemplo, un diagrama de barras, se puede utilizar el paquete `pgfplots` que se basa en `pgf/tikz` y simplifica su construcción. Para ello es necesario incluir en el preámbulo: `\usepackage{pgfplots}`, que añade la librería `pgfplots`. Por ejemplo:



Se obtiene con este código:

```
\begin{tikzpicture}
\begin{axis}[
xbar, xmin=0,
width=12cm, height=6cm, enlarge y limits=0.2,
xlabel={Lenguajes de programación (repositorios activos)},
symbolic y coords={Javascript, Java, Python, CSS, PHP, Ruby,
C++},
```

```

nodes near coords, nodes near coords align={horizontal},
]
\addplot coordinates {(323928, Javascript)
(222852, Java)
(164852, Python)
(164585, CSS)
(138771, PHP)
(132848, Ruby)
(86505, C++) };
\end{axis}
\end{tikzpicture}

```

Calendarios[\[editar\]](#)

Existe un paquete basado en TikZ que permite gestionar días, días de la semana, meses y años, y dibujarlos de forma sencilla. Para utilizarlo, se incluye `\usetikzlibrary{calendar}`. Por ejemplo:

Abril							
				1	2	3	
4	5	6	7	8	9	10	2
11	12	13	14	15	16	17	9
18	19	20	21	22	23	24	16
25	26	27	28	29	30		23
							30

Se genera con el siguiente código:

```
\begin{tikzpicture} [column sep=1cm]
```

```

\matrix {
\calendar[dates=2016-04-01 to 2016-04-last,week list, month
label above centered];
&
\calendar[dates=2016-05-01 to 2016-05-last,week list, month
label above centered];

};
\end{tikzpicture}

```

Para que el texto de los meses salga en español es necesario incluir en el preámbulo, lo siguiente: `\usepackage[spanish]{translator}`. Justo después de cargar el paquete babel.

Referencias^{[[editar](#)]}

1. Página del proyecto [TikZ](#) en CTAN.
2. Introducción simple a TikZ : [A very minimal introduction to TikZ](#).
3. Página del paquete [\[https://www.ctan.org/pkg/pgfplots\]](https://www.ctan.org/pkg/pgfplots) `pgfplots`.

- [Volver arriba](#)[↑] El círculo con el cuadrado inscrito.
- [Volver arriba](#)[↑] *Path* , en inglés.

Manual de LaTeX/La estructura de un documento en LaTeX

< [Manual de LaTeX](#)

Ya se ha explicado cómo se compila un documento en `LaTeX` , pero no se ha hablado aún de cómo escribir el documento a compilar. En este capítulo se analiza la estructura básica de un documento, y en el siguiente capítulo se expondrán los conceptos básicos sobre la escritura

de texto en `LaTeX` .

Sumario

[ocultar]

- [1La estructura](#)
 - [1.1Preámbulo](#)
 - [1.1.1Paquetes](#)
 - [1.2Cuerpo](#)

- 1.2.1 Órdenes o macros
- 1.2.2 Principios básicos de la escritura

La estructura [\[editar\]](#)

La estructura de un documento en `LaTeX` se divide en dos grandes partes: el preámbulo y el cuerpo del texto. El siguiente ejemplo muestra un documento mínimo apropiado para el español:

```
\documentclass[spanish]{article}
\usepackage{babel}
\usepackage[T1]{fontenc}
\usepackage{textcomp}
\usepackage[utf8]{inputenc} % Puede depender del sistema o editor
\begin{document}
Aquí iría el texto del documento en sí.
\end{document}
```

Preámbulo [\[editar\]](#)

En el *preámbulo* se escriben las instrucciones fundamentales que indican a `LaTeX` qué [clase de documento](#) se va a escribir y qué características va a tener éste, así como también las que

indican a `LaTeX` qué [paquetes](#) se deben cargar. El preámbulo siempre empezará con la instrucción:

```
\documentclass[<opciones>]{<plantilla_documento>}
```

Para definir la plantilla que se va a emplear en el documento, como por ejemplo `article` o `report`, que determinan diferentes estilos. En general, los argumentos que toma este comando son las llamadas *clases* de documento, y pueden aceptar diferentes opciones. Por ejemplo, la instrucción:

```
\documentclass[12pt, letterpaper]{book}
```

Declara que el documento es un libro, con el tamaño de letra configurado a 12 puntos y utilizando papel tamaño carta. En vez de `letterpaper`, se pueden usar otros tamaños de papel, como lo es `A4` (`a4paper`).

Paquetes [\[editar\]](#)

Se llama paquete a una extensión del sistema básico que añade nuevas funciones. Hay, literalmente, cientos de paquetes con muy diversas funciones: inserción de imágenes (`graphicx`), paquetes gráficos (`TikZ`), internacionalización (`babel`, `polyglossia`), color (`xcolor`), música, ajedrez, ediciones críticas, secuencias de aminoácidos, etc. Todos estos paquetes deberán ser declarados con:

```
\usepackage[<opciones>]{<paquete>}
```

Donde entre los corchetes estará el nombre del paquete a usar, por ejemplo:

```
\usepackage{amssymb}
```

Para cargar el paquete `amssymb`, que proporciona símbolos matemáticos de la American Mathematical Society. Si una clase de documento o paquete que queremos cargar ofrece opciones y nosotros no especificamos la que queremos, se cargarán las opciones por defecto.

Cuerpo[\[editar\]](#)

El cuerpo del documento consiste en prácticamente todo lo que aparecerá en nuestra compilación. Es aquí, pues, donde escribiremos el texto verdadero. Comienza con la instrucción

```
\begin{document}
```

y termina con:

```
\end{document}
```

Todo lo que se escriba con posterioridad a esta instrucción será ignorado por `LaTeX` y no se compilará.

Una vez que iniciemos el cuerpo del documento debemos escribir al final de todo lo escrito la instrucción de cierre `\end{document}` aunque no hayamos terminado todo el documento, pues de otra manera tendremos un error en el proceso de la compilación y no podremos ir viendo cómo van quedando nuestros avances.

Órdenes o macros[\[editar\]](#)

Hay ciertos caracteres que tienen una función especial, como por ejemplo, el símbolo `\`, por el que comienzan todas las instrucciones, o las llaves y corchetes, que contienen los datos y opciones de las instrucciones. Por ejemplo, en `\title{Un título}` hay una orden (en muchos casos también llamadas macros), que es `\title` y que ajusta el título del documento con el dato (o argumento) que sigue, que es `Un título` (en este punto no se añade realmente el título al documento, sino que `LaTeX` tan sólo lee el dato y lo guarda para cuando haga falta).

Otro detalle que hay que destacar es que las órdenes que no van seguidas de algún argumento descartan el o los espacios que le siguen (con la excepción de las órdenes que consisten en un símbolo, como `\#`, `\%` o `\$`). Aunque hay pocas órdenes de este tipo —la mayoría tienen algún dato o consisten en un símbolo— es muy importante tener esto presente por si se diera el caso. La solución en tales casos suele pasar por añadir un «dato» vacío, es decir, un par de llaves.

Finalmente, hay que señalar que algunas órdenes no van seguidas de uno o varios argumentos, sino que operan sobre el texto que le sigue hasta que termina el bloque actual delimitado por llaves. En ocasiones, incluso, hay dos variantes que funcionan de cada uno de estos modos. En caso simple es el siguiente:

```
\textbf{texto en negrita}
{\bfseries texto en negrita}
```

Estas dos formas son equivalentes, pero por lo general se prefiere el primer tipo.

Un tipo especial de orden es la que delimita un bloque del documento. Van siempre por pares:

```
\begin{...}
...
\end{...}
```

Una estructura así se llama *entorno* o *ambiente* y la más importante es justamente la que abarca el cuerpo del documento.

Principios básicos de la escritura[\[editar\]](#)

Todo bloque de texto separado del resto con líneas en blanco se considera un párrafo. No es el único caso en que LaTeX considera que hay un párrafo, pero sí es el más importante. En este caso, se lee el texto contenido en ese bloque y LaTeX lo procesa con objeto de encontrar las mejores divisiones de línea, los mejores guiones y el mejor espaciado posible para el párrafo. También se preocupa de encontrar el mejor punto para cambiar de página, así como de cuadrar el resultado en la página. Todo ello, naturalmente, sin necesidad de intervención directa de quien escribe.

El texto de cada párrafo se escribe de modo normal, con algunas salvedades importantes. En primer lugar, un espacio entre palabras vale lo mismo que dos, tres o cientos, siempre que no se deje una línea en blanco. De esta forma se evitan espaciados irregulares que en sistemas WYSIWYG aparecen en ocasiones al teclear por error dos espacios seguidos. En segundo lugar, LaTeX proporciona un buen número de caracteres adicionales a menudo inexistentes en los teclados y que se pueden introducir como órdenes; por ejemplo, `\textdagger` inserta una cruz (†) en el punto donde aparece. Véamoslo con un ejemplo de documento completo:

```
\documentclass[spanish]{article}
\usepackage{babel}
\usepackage[T1]{fontenc}
\usepackage{textcomp}
\usepackage[utf8]{inputenc} % Puede depender del sistema o editor

\title{Un título}
\author{El autor}
\date{5 de marzo del 2015}

\begin{document}
```

Un breve texto introductorio que servirá como ejemplo para mostrar qué forma tiene un párrafo. Hasta ahora solo tenemos uno, que concluimos con una línea en blanco.

Tras la línea en blanco, tenemos otro párrafo. En él, además, escribiremos una cruz (`\textdagger{}`).

Pero interrogaciones, comillas, etc., se escriben normalmente: ¿de verdad?, «comillas», ¡qué bien!

Los estilos de letra también se introducen con órdenes, como `\textit{cursiva}` y `\textbf{negrita}`.

```
\end{document}
```

Manual de LaTeX/La estructura de un documento en LaTeX/Cuerpo/Estilos de página

< [Manual de LaTeX](#) | [La estructura de un documento en LaTeX](#)

La numeración de páginas y la impresión de encabezados en las mismas constituyen el estilo de la página. Cambios en el estilo de página pueden realizarse con el comando

```
\pagestyle{'estilo'}
```

Los estilos que ofrecen las clases de documento estándar

de `empty`, `plain` y `headings`. Con `\pagestyle{empty}` hacemos que las páginas queden sin número de página ni encabezado; con `\pagestyle{plain}`, que es el estilo por defecto, obtenemos páginas numeradas, pero sin encabezado; con `\pagestyle{headings}` obtenemos páginas numeradas y con encabezado. Más específicamente, `\pagestyle{headings}` produce efectos distintos según la clase de documento y las opciones que para ella se especifiquen. Por ejemplo, con la clase `article`, `\pagestyle{headings}` nos dará el número de página al pie y un encabezado con el nombre de la sección, y si hemos elegido la opción `twoside`, el

encabezado será el nombre de la sección en las páginas pares y el nombre de la subsección en las páginas impares. Para el caso de la clase `book`, `\pagestyle{headings}` pondrá el número de página en la parte exterior de la cabecera (lado izquierdo en páginas pares y lado derecho en páginas impares) y el encabezado (que será el nombre del capítulo en páginas pares y el nombre de la sección en páginas impares) en la parte interior de la cabecera.

Si queremos cambiar el estilo de una página en particular, usamos

```
\thispagestyle{'estilo'}
```

que toma los mismos valores que `\pagestyle{}`.

Para especificar por nuestra propia cuenta que es lo que aparecerá en la cabecera, podemos usar la instrucción

```
\pagestyle{myheadings}
```

que pondrá los encabezados según estos estén indicados con los comandos

```
\markboth{'encabezado izquierdo'}{'encabezado derecho'}
```

y

```
\markright{'encabezado derecho'}
```

Notar que con la opción de clase `oneside`, los encabezados sólo pueden ser los derechos (pues no hay páginas que estén a la izquierda).

Al utilizar el estilo `\pagestyle{headings}`, vemos que la letra del encabezado aparece en caracteres inclinados y en mayúsculas. Esto se debe a que las clases de documento estándar

de así lo definen. Para reajustar los encabezados y los pies, en lo que respecta a las mayúsculas y en otros detalles, hay dos paquetes útiles: `fancyhdr` y `titleps`. Un ejemplo simple de este último es:

```
\newpagestyle{main}{
  \sethead[\thepage][\chaptertitle][(\thesection] % pares
           {\thesection)}{\sectiontitle}{\thepage}} % impares
\pagestyle{main}
```