

L^AT_EX para infomates

Resumen

Este documento pretende ser un manual de uso de L^AT_EX y de los paquetes personalizados que tenemos para escribir los [apuntes de matemáticas e informática](#). La idea es que sirva para aprender L^AT_EX básico (capítulo [I](#)), que introduzca aspectos de L^AT_EX avanzado (capítulo [II](#)) y que documente los paquetes y comandos personalizados que hemos ido desarrollando a lo largo del tiempo (capítulo [III](#)). Al final del documento (pág. [45](#)) hay un índice alfabético de los comandos y conceptos tratados, para buscar rápidamente algún tema concreto en el texto.

Índice general

I	Una introducción a LaTeX	3
I.1	Hola Mundo: Primer documento con LaTeX	4
I.1.1	Instalando un entorno LaTeX	4
I.1.2	Un primer documento	4
I.2	Formato y maquetación	6
I.2.1	Dando estructura: secciones, subsecciones, párrafos y más	6
I.2.2	Dando formato: negritas, cursivas y demás tipo de letra	7
I.2.3	Entornos menos básicos: listas, tablas, imágenes y más	8
I.2.3.1	Listas	8
I.2.3.2	Imágenes	8
I.2.3.3	Tablas	9
I.2.3.4	Notas al pie	10
I.2.3.5	Etiquetas y referencias	10
I.3	Matemáticas en LaTeX	11
I.3.1	Subíndices y superíndices, fracciones y raíces	12
I.3.2	Matrices	12
I.3.3	Delimitadores	12
I.3.4	Símbolos, operadores y texto	13
I.3.5	Caracteres en otras fuentes	14
I.3.6	Entornos matemáticos	14
I.4	Preguntas frecuentes	15
II	LaTeX avanzado	17
II.1	¿Cómo funciona LaTeX?	17
II.2	LaTeX para usuarios avanzados	18
II.2.1	¿Dónde están documentados los paquetes?	18
II.2.2	Cambiando los márgenes del documento	19
II.2.3	Manejando documentos grandes con varios archivos	19
II.2.4	Cómo personalizar la tabla de contenidos y la numeración de secciones	19
II.2.5	Referencias todavía más automáticas: fancyref	20
II.2.6	Estilos de página: pies y cabeceras	21
II.2.7	Dibujos y gráficas de funciones con código LaTeX: Tikz	22
II.2.8	Un compilador más completo: latexmk	25
II.2.9	Beamer: presentaciones con LaTeX (o cómo dije adiós a Powerpoint)	26
II.3	Ampliando LaTeX: Desarrollo de comandos, entornos, paquetes y clases	26
II.3.1	Comandos y entornos	27
II.3.1.1	Manejo avanzado de argumentos	28
II.3.1.2	Comandos de bajo nivel TeX y expansión de argumentos	29
II.3.2	Paquetes y clases	30

⁰ Documento compilado el 7 de mayo de 2016 a las 12:37

II.3.3 Paquetes auxiliares para el desarrollo	31
III Manual paquetes extendidos	32
III.1 Introducción: ¿qué podemos hacer con los paquetes extendidos?	32
III.2 Estructura de los paquetes extendidos	32
III.3 Documentación de los paquetes	33
III.3.1 <i>exmath.sty</i> : Comandos y entornos adicionales de matemáticas . .	33
III.3.1.1 Entornos para teoremas y similares	33
III.3.1.2 Definiciones y conceptos	34
III.3.1.3 Ejercicios	34
III.3.1.4 Imágenes	35
III.3.1.5 Comandos adicionales	36
III.3.2 <i>fancysprefs.sty</i> : Referencias mejoradas	36
III.3.3 <i>MathUnicode.sty</i> : Letras griegas directamente desde el teclado . .	36
III.3.4 <i>tikztools.sty</i> : Herramientas y comandos extra para Tikz	36
III.3.5 <i>fastbuild.sty</i> : Recortando los tiempos de compilación	39
III.3.6 <i>apuntes.cls</i> : La base para crear documentos de apuntes	39
III.3.6.1 Cabecera de páginas	39
III.3.6.2 Portada	39
III.3.7 <i>ejercicios.cls</i> : documentos simples y entrega de ejercicios	40
A Comandos	41
Índice alfabético	45

Capítulo I

Una introducción a L^AT_EX

Vayamos primero con las preguntas básicas: ¿qué es L^AT_EX? Es un sistema de generación de documentos¹. En otras palabras, nosotros escribimos cosas y L^AT_EX nos genera un documento como este mismo manual.

La siguiente pregunta es: ¿por qué usar L^AT_EX y no Word o LibreOffice? Esta pregunta ya no tiene una respuesta tan clara, aunque podemos comentar algunas ventajas. Por ejemplo, es mucho más fácil obtener documentos maquetados a la perfección: L^AT_EX nos permite despreocuparnos del formato mientras escribimos porque él se encarga de todo: gestión de los párrafos, tamaño de fuente² para cada posible situación, colocación de los imágenes, saltos de página, numeración... Básicamente, todo lo que se pueda automatizar se automatiza, mientras que nosotros nos preocupamos de lo importante: escribir. Además, no hace falta saber cómo maquetar bien los documentos para que nos queden bien, y además la mejor forma de hacer algo es siempre la más fácil.

Por supuesto, esto viene con un coste, y es que L^AT_EX no es fácil. Sólo hay que ver que estás leyendo un manual para aprender: ¿cuándo has leído un manual para usar Word?

La razón por la que no es fácil es que con L^AT_EX no escribimos letras, pulsamos botones y ya. En L^AT_EX lo que se hace es escribir comandos y compilar el documento para generar algo legible, así que podemos decir que se parece bastante a programar. Por ejemplo, cuando queremos poner algo en negrita, no nos vamos al botón correspondiente en el programa: lo que hacemos es escribir el comando correspondiente (`\textbf{negr ita}`) para que cuando el compilador pase por ahí, sepa que tiene que poner esas palabras en negrita.

Esto tiene una cosa mala, y es que escribir un documento *.tex* puede llegar a ser un poco infernal, porque ves los comandos y no lo que producen. Otro problema es que, al escribir comandos, puedes equivocarte y que el documento no compile. No sería un problema demasiado importante si no fuese porque la mitad de las veces los errores que muestra un compilador son complementamente inútiles. Más tarde veremos cómo enfrentarnos a esos errores de compilación, pero primero empezaremos por lo más interesante: nuestro primer documento L^AT_EX.

¹En inglés, *document typesetting system*, que queda mejor.

²A modo de curiosidad sobre las fuentes, una de las razones por las que L^AT_EX saca documentos tan “profesionales” es lo que se llama el *font kerning* o espaciado entre letras. Por si alguien quiere saber más, [este artículo en inglés es bastante interesante](#).

I.1. Hola Mundo: Primer documento con L^AT_EX

Decíamos antes que L^AT_EX necesita un compilador, así que antes de nada vamos a necesitar instalarlo. También instalaremos un editor especializado. Aunque no es necesario (los documentos *.tex* son texto plano que se pueden editar hasta con el bloc de notas), sí que nos vendrá muy bien para empezar, ya que nos facilitan mucho la vida con una interfaz amable y botones que hacen las cosas por nosotros.

I.1.1. Instalando un entorno L^AT_EX

Lo que necesitaremos bajarnos en nuestro ordenador será una *distribución* L^AT_EX, que es una colección de programas y archivos de configuración que nos permitirán generar nuestros documentos. Según el sistema que usemos, tendremos que bajarnos una cosa u otra:

- **Windows.** Antes de decir cómo descargarse L^AT_EX, haré una recomendación, y es que Windows no es el sistema más apropiado para esto, sobre todo si queremos editar los apuntes de infomates. Es mejor irse a un sistema POSIX (Linux o Mac OS X) que tenga una consola decente y con *bash* para poder ejecutar los *scripts* que tenemos en el repositorio. Además, estudiando informática, se acaba usando Linux sí o sí, así que cuanto antes se empieza a usarlo mejor. Si a pesar de todo queremos seguir con Windows, lo que hay que bajarse es [la distribución MikTeX](#) e instalarla (es un instalador normal y corriente).
- **Mac OS X.** Hay varias formas de instalarse L^AT_EX en OS X, aunque yo recomiendo usar Macports³ e instalar los paquetes necesarios con `sudo port install texlive-basic texlive-bin-extra texlive-fonts-extra texlive-lang-spanish texlive-latex texlive-latex-extra texlive-math-extra`.
- **Linux basado en Debian (Ubuntu y similares).** El paquete básico para usar L^AT_EX es *texlive*, así que con `sudo apt-get install texlive` debería bastar. Es recomendable bajarse además *texlive-latex-base*, *texlive-latex-extra*, *texlive-latex-recommended* para paquetes adicionales que no están en la distribución base.
- **Linux no basado en Debian.** Como de estos hay muchos y los que se lo instalan suelen ser un poco más apañados que la media, daré por supuesto que sabrán encontrar los paquetes que necesitan instalar sabiendo los que necesita Ubuntu.

Una vez que tenemos L^AT_EX instalado, nos bajaremos el editor. Personalmente, recomiendo [TeXStudio](#)³, que funciona bastante bien, es fácil de usar y además es bastante potente.

I.1.2. Un primer documento

Una vez que tenemos todo instalado, podemos hacer nuestro primero documento. El archivo está [disponible para que se pueda descargar y bajar](#) sin tener que copiar y pegar de un PDF.

³Macports es un gestor de paquetes al estilo *apt-get* para OS X. [Estas son las instrucciones para instalarlo.](#)

```

1 \documentclass[a4paper]{article}
2
3 % Las líneas que empiezan por % son comentarios
4
5 \usepackage[utf8x]{inputenc} % Esto es para que LaTeX reconozca tildes
6 \usepackage[spanish]{babel} % Esto para que las cadenas estén en español (por ejemplo,
   ↳ la fecha)
7 \usepackage{amsmath} % Esto nos permite poner fórmulas matemáticas
8
9 \title{Hola mundo}
10 \author{Yo}
11 \date{\today}
12
13 \begin{document}
14 \maketitle
15
16 \section{Mi primera sección}
17
18 \subsection{Mi primera subsección}
19
20 \subsubsection{Mi primera subsubsección}
21
22 \paragraph{Mi primer párrafo} 0, de cómo me di cuenta de que no hay más niveles de
   ↳ sección.
23
24 Este es mi segundo párrafo, así que voy a aprovechar para poner \textbf{una negrita},
   ↳ una \textit{cursiva} y hasta un poco de texto en \texttt{monoespacio}\footnote{0 en
   ↳ modo typewriter, y así aprovecho para poner una nota al pie automáticamente}.
25
26 \begin{itemize}
27   \item También puedo poner listas
28   \item Como esta
29 \end{itemize}
30
31 Y además, puedo escribir matemáticas con  $1 + 1 = 2$  o, para que vayan en su propia
   ↳ línea, con \[ \int f(x) = 32 + \frac{3}{4} \cdot 1 + \dotsb + 3^2 + a_3 + b_{ij} \]
32
33 A modo de curiosidad, podemos ver que para \LaTeX\ muchos espacios
   ↳ en blanco son lo mismo que uno sólo.
34 De hecho,
35 un salto de línea
36 sin una línea blanca en medio no genera un nuevo párrafo en el documento.
37
38 Ahora sí que tenemos un nuevo párrafo.
39 \end{document}

```

Es recomendable toquetear el documento y ver qué pasa si quitamos comandos, o si cambiamos letras. La figura I.1 muestra cómo queda este documento si lo vemos en TeXStudio.

En este documento podéis ver varios comandos (los que empiezan por “\”), de los cuales muchos no merecen demasiada explicación. Otros los veremos más tarde, como los relacionados con ecuaciones matemáticas.

Lo que sí vamos a comentar son los comandos `\documentclass`, `\usepackage` y los `\begin{document}` y `\end{document}`. El primero debería estar siempre en la primera línea de cualquier documento L^AT_EX, y le dice al compilador qué tipo de documento estamos escribiendo. En este caso es *article*, aunque podría ser cualquier otro como *book* o *report*. El tipo de documento definirá qué márgenes usamos, el tipo de letra, los tipos de secciones que se pueden usar o los paquetes que se incluyen por defecto.

Eso nos lleva al siguiente comando, `\usepackage`. Como su nombre indica, carga un

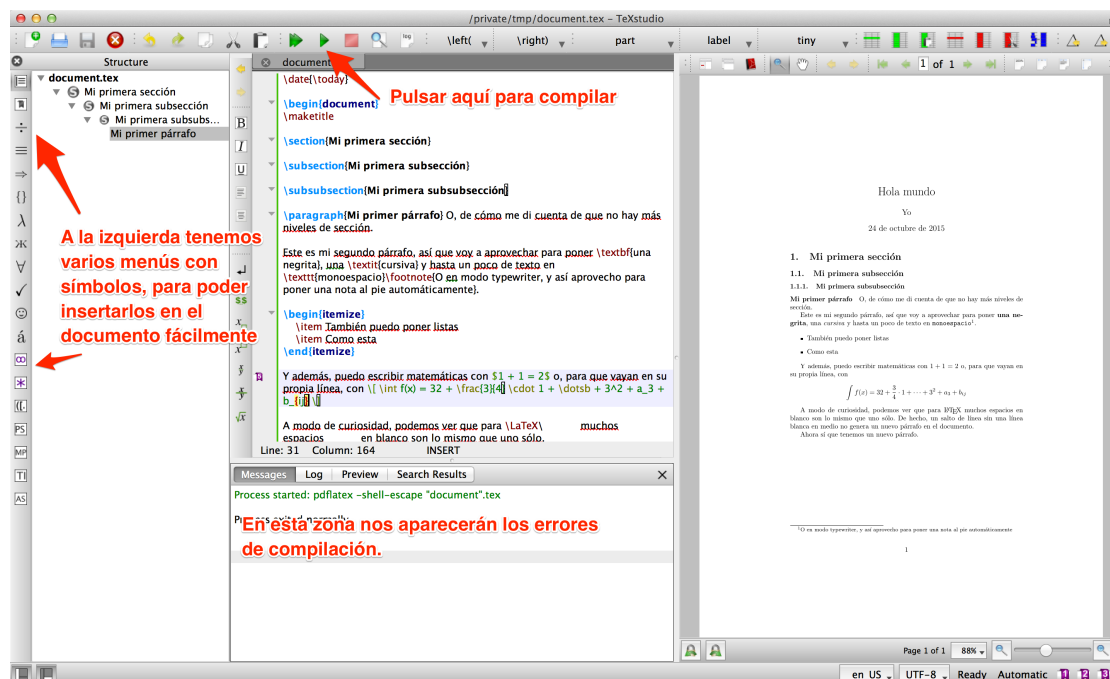


Figura I.1: Una captura de pantalla de TeXStudio con el documento de muestra, junto con algunas indicaciones sobre la interfaz.

paquete en el documento y siempre van en el preámbulo (ahora veremos qué es eso). Un paquete de LaTeX puede o bien proporcionarnos comandos (por ejemplo, *amsmath* nos proporciona los comandos `\int` o `\dotscsb`) o bien cambiar el comportamiento de LaTeX, como *babel*, que con la opción *spanish* traducirá las cadenas que el compilador inserte en el documento (por ejemplo, cuando ponga la fecha automáticamente o cuando ponga “Figura N” en las etiquetas de imágenes) en español.

Por último, nos quedan los comandos `\begin{document}` y `\end{document}`. Son un ejemplo de lo que se llaman “entornos” en LaTeX. Como uno podría imaginar, marcan el inicio y el final del documento en sí. Lo que esté dentro de estos dos comandos es lo que se ve en el documento final. Lo que está fuera (antes del `\begin{document}`, porque después de `\end{document}` no debería haber nada) es lo que se llama el preámbulo, que podríamos decir que es la sección de “configuración” del documento.

I.2. Formato y maquetación

Ahora que ya sabemos a grandes rasgos cómo funciona un documento LaTeX, vamos a ir viendo explicaciones más en detalle de cada aspecto, empezando por lo básico: cómo dar formato y maquetar nuestro documento.

I.2.1. Dando estructura: secciones, subsecciones, párrafos y más

Normalmente, los documentos se organizan en secciones, subsecciones y similares. LaTeX nos lo pone muy fácil para hacerlo. Sólo necesitaremos poner el comando correspondiente y se encargará de ponerle el número que le corresponda y de añadirlo a la tabla de contenidos si la hemos activado. Para que la tabla de contenidos nos aparezca en el documento, sólo tenemos que escribir `\tableofcontents` donde queramos que se

muestre.

Las posibles variantes de sección son las siguientes. Como aparece en el comentario, el comando `\chapter` sólo se puede usar en algunos tipos de documento, como *report* o *book* (no *article*).

```
\chapter{Un capítulo} % Sólo en algunos tipos de documento
\section{Una sección}
\subsection{Una subsección}
\subsubsection{Una subsubsección}
```

```
\paragraph{Un párrafo}
```

```
\subparagraph{Un subpárrafo}
```

Además, como indicábamos en el documento de muestra, LaTeX es algo especialito con los espacios en blanco y los párrafos. Varios espacios en blanco es lo mismo que uno sólo, y los saltos de línea se ignoran salvo que haya una línea en blanco, que es lo que indica a LaTeX que hay un nuevo párrafo.

Si se quiere forzar un salto de línea (no recomendado, LaTeX sabe mejor que cualquiera de nosotros cómo y donde romper las líneas; y si quieres cambiar de párrafo deberías poner una línea en blanco) podemos usar `\\`, `\newline` o `\pbreak`.

```
1 Esta línea
2 y esta otra
3 son del mismo párrafo.
4
5 Ahora si cambiamos de párrafo, y tambien
   podemos ver que
6 los     espacios dan igual, con uno
   basta.
7
8 El cambio de línea forzado queda un poco
   mas raro y \\
9 no cambia bien de
10 párrafo. Es mejor no usarlo.
```

Esta línea y esta otra son del mismo párrafo.

Ahora si cambiamos de párrafo, y tambien podemos ver que los espacios dan igual, con uno basta.

El cambio de línea forzado queda un poco mas raro y no cambia bien de párrafo. Es mejor no usarlo.

I.2.2. Dando formato: negritas, cursivas y demás tipo de letra

Una revisión rápida por los posibles tipos de letra que podemos tener:

```
1 \textbf{negrita}, \textit{cursiva}, \texttt{monoespacio}
2
3 \Huge{muy enorme}, \huge{enorme},
4 \LARGE{muy muy grande},
5 \Large{muy grande}, \large{grande},
6 \normalsize{normal}, \small{reducido},
7 \tiny{muy reducido}.
```

negrita, *cursiva*, monoespacio
muy **enorme**,
enorme, muy muy gran-
de, muy grande, grande, normal,
reducido, muy reducido.

También podemos querer escribir código y no preocuparnos porque LaTeX lo trate de interpretar. Para eso podemos usar los entornos *verbatim*:


```

1 En el texto, podemos usar
2 \verb|\micomando| (atentos
3 a las barras verticales para
4 delimitar el argumento)
5 para que \LaTeX{}
6 no trate de interpretar lo
7 que ponemos como un comando.
8
9 Si queremos algo mas extenso
10 podemos usar el entorno
11 \textit{verbatim}:
12
13 \begin{verbatim}
14 \textit{LaTeX no interpreta esto}
15 \end{verbatim}

```

En el texto, podemos usar `\micomando` (atentos a las barras verticales para delimitar el argumento) para que `LaTeX` no trate de interpretar lo que ponemos como un comando.

Si queremos algo mas extenso podemos usar el entorno *verbatim*:

```
\textit{LaTeX no interpreta esto}
```

I.2.3. Entornos menos básicos: listas, tablas, imágenes y más

Pasamos ahora a temas básicos, pero algo menos básicos que poner formatos sencillos. Ahora toca ver cómo meter listas, imágenes, tablas, notas al pie y referencias en nuestros documentos.

I.2.3.1. Listas

Las listas son sencillas, y lo mejor es verlo con ejemplos. Por supuesto, las listas se pueden anidar.

```

1 \begin{itemize}
2 \item Uno
3 \item Dos
4 \end{itemize}
5
6 \begin{enumerate}
7 \item Numerado
8 \item Por defecto
9 \begin{enumerate}
10 \item Incluso con anidaciones
11 \item Podemos poner todos los niveles
12 \begin{enumerate}
13 \item Que queramos
14 \end{enumerate}
15 \item \LaTeX{} hace todo
16 \end{enumerate}
17 \item De forma automatica.
18 \end{enumerate}
19
20 \begin{description}
21 \item[Etiqueta] Elemento
22 \end{description}

```

- Uno
 - Dos
1. Numerado
 2. Por defecto
 - a) Incluso con anidaciones
 - b) Podemos poner todos los niveles
 - 1) Que queramos
 - c) `LaTeX` hace todo
 3. De forma automatica.

Etiqueta Elemento

I.2.3.2. Imágenes

Para poner imágenes, sólo es necesario un comando, `\includegraphics`. La cuestión es que eso nos inserta la imagen directamente, y no nos permite poner una etiqueta ni referenciarlo más tarde. Lo que haremos será encerrar ese comando en un entorno *figure*:

```

1 \begin{figure}[hbt]
2 \centering
3 \includegraphics[width=0.4\textwidth]{
  Patata.jpg}
4 \caption{Etiqueta para la figura}
5 \label{fig:FiguraEjemplo}
6 \end{figure}

```



El entorno *figure* es lo que se llama un entorno flotante. La ventaja es que nos da opciones adicionales y además le dice a L^AT_EX que coloque la imagen donde sea más conveniente, evitando dejar muchos espacios en blanco o rompiendo el flujo del texto. Esta es una de las cosas que más cambian con respecto a Word, y de lo que más cuesta convencer: es mejor dejar que L^AT_EX ponga la imagen donde quiera: normalmente será una posición mejor que la que podamos poner nosotros, y además nos evitaremos dolores de cabeza cuando cambiemos el texto y se nos descuadren todas las imágenes (como en Word).

Lo que sí podemos decirle a L^AT_EX es donde preferimos que ponga la imagen (luego nos hará caso o no) con las opciones `[hbt]`. Cada letra denota una preferencia, y van ordenadas según su prioridad: primero intenta poner la imagen donde la has escrito; si no puede, en la parte inferior de la página; si no, en la parte superior; y si ahí tampoco puede la dejará en una página dedicada a imágenes. Las letras se pueden quitar y cambiar de orden según querramos que aparezca la imagen.

El comando `\includegraphics[width=X\textwidth]{ruta/a/la/imagen}` es el que inserta la imagen en sí. Normalmente solemos limitar el ancho de la imagen con la opción `[width=X\textwidth]`, donde *X* es el porcentaje (entre 0 y 1) del ancho del texto que queremos que ocupe. Por ejemplo, `0.5\textwidth` hará que la imagen ocupe la mitad del texto de ancho.

El comando `\caption` nos permite ponerle una etiqueta o “caption” a la imagen, para explicar qué es. Es conveniente usarlo, porque siempre se mantiene pegado a la imagen y además nos permite ver qué índice tiene la figura. Por último, el comando `\label` le asigna un identificador a la figura para que podamos referenciarla más tarde (veremos cómo hacer eso en la sección [I.2.3.5](#)).

I.2.3.3. Tablas

L^AT_EX también nos permite incluir tablas en los documentos, aunque no es especialmente cómodo. Necesitaremos usar el entorno *tabular*, al que le especificamos las columnas que queremos usar. Después, ponemos los contenidos de la tabla, separando cada celda con el carácter `&` y cada fila con el salto de línea `\\`. Veamos un ejemplo:

```

1 \begin{tabular}{r|l|r|c|l}
2 0 & 1 & 2 & 3 \\ \hline
3 texto & texto & texto & texto
4 \end{tabular}

```

0	1	2	3	
texto	texto	texto	texto	

El primer argumento de *tabular* es el especificador de columnas. Normalmente son los caracteres *r*, *l*, *c*, que indican alineación de la celda a la derecha, a la izquierda o al centro respectivamente; y separados (o no) por caracteres `|`, que le dicen a L^AT_EX

que ponga una barra vertical entre esas dos columnas. Para poner barras horizontales, ponemos el comando `\hline` al final de la línea.

Al igual que con las imágenes, la mayor parte de las veces nos interesará poner la tabla en un entorno flotante, *table*, para que L^AT_EX la coloque donde mejor venga y podamos poner un *caption* y la referencia con *label*.

```
1 \begin{table}[hbt]
2 \centering
3 \begin{tabular}{r|l|r|c|l}
4 0 & 1 & 2 & 3 & \\ \hline
5 texto & texto & texto & texto & 
6 \end{tabular}
7 \caption{Etiqueta para la tabla}
8 \label{tbl:TablaEjemplo}
9 \end{table}
```

0	1	2	3	
texto	texto	texto	texto	

Tabla I.1: Etiqueta para la tabla

I.2.3.4. Notas al pie

Para poner notas al pie, simplemente tenemos que poner el comando `\footnote{texto de la nota}` en medio del texto. Con eso, L^AT_EX se encargará de colocar la nota en la página que toque y de enlazarla convenientemente. Si tenemos una nota que es demasiado larga y no queremos ponerla en medio del texto, podemos poner simplemente `\footnotemark` donde queremos que aparezca el numerito, y después usar `\footnotetext{texto}` tranquilamente.

Ponemos un ejemplo de uso, aunque la salida no queda del todo bien por cosas de L^AT_EX. Es una pequeña muestra de que a veces las notas al pie son un poco puñeteras y en ciertos entornos no salen. De tofas formas, no deberíamos encontrar ningún problema si las usamos normalmente en el texto.

```
1 En medio del texto \footnote{
2 El texto es esto} podemos poner
3 una nota al pie. Tambien podemos
4 poner \footnotemark y llenarlo
5 mas tarde.
6
7 \footnotetext{Por ejemplo, aqui lo
8 ponemos porque es una nota muy
9 larga y no queremos ensuciar el
10 texto.}
```

En medio del texto podemos poner una nota al pie. Tambien podemos poner⁴ y llenarlo mas tarde.

I.2.3.5. Etiquetas y referencias

Una característica muy potente de L^AT_EX son las referencias. Usa `\label{nombreEtiqueta}` cuando quieras referenciar otras partes del documento con `\ref{nombreEtiqueta}`. Por ejemplo, si pones una etiqueta debajo de un comando de sección, `\ref{nombreEtiqueta}` mostrará el número de esa sección. También puedes hacerlo en ecuaciones, figuras, tablas, listados de código y otros entornos definidos en el paquete `exmath`. Para figuras y tablas, concretamente, hay que poner el `\label` justo después del `\caption`, tal y como aparece en las secciones I.2.3.2 y I.2.3.3.

```

1 Ponemos una ecuacion con su etiqueta:
2 \begin{equation}
3 1 + 1 = 2 \label{eq:SuperImportante}
4 \end{equation}
5
6 Ahora podemos referenciarla: la
7 ecuacion \ref{eq:SuperImportante}.
8
9 Tambien podemos hacerlo con \eqref{eq:
   SuperImportante}.

```

Ponemos una ecuacion con su etiqueta:

$$1 + 1 = 2 \quad (\text{I.1})$$

Ahora podemos referenciarla: la ecuacion [I.1](#).

Tambien podemos hacerlo con [\(I.1\)](#).

La principal ventaja de usar las referencias así es que no tenemos que preocuparnos más de ellas: da igual que cambie la posición de lo que estamos referenciando, que su numeración sea distinta o incluso que cambiemos el modo de numerar (por ejemplo, que en lugar de usar números usemos letras): L^AT_EX se encarga de mantenerlo todo coherente.

Lo único que tendremos que acordarnos de hacer es de compilar bien hasta que las referencias se resuelvan: a veces, en la primera compilación L^AT_EX no encuentra las referencias y hay que hacer más pasadas para que funcione (ver la sección [II.1](#) para una explicación algo más extensa de esto).

I.3. Matemáticas en L^AT_EX

Habiendo llegado hasta aquí, ya deberíamos de saber generar documentos básicos en L^AT_EX. Ahora bien, nos falta una de las características más potentes de este sistema, y que siendo infomates debería interesarnos mucho: escribir matemáticas.

L^AT_EX tiene lo que se llama un “modo matemáticas”, donde dejará de interpretar texto e interpretará los símbolos que pongamos como lo que son: ecuaciones y fórmulas matemáticas. Los principales son dos, aunque hay bastantes más. Por un lado, los dólares (\$) nos permiten poner ecuaciones en línea, en el párrafo. Por otro, con `\[. . \]` L^AT_EX nos pondrá la ecuación en una línea separada. Veamos un ejemplo.

```

1 Podemos poner ecuaciones $1 = 2$
2 en linea, o tambien separadas con
3 \[ 1 = 2 + 4 \]
4
5 No hace falta poner el salto de
6 linea, se hace solo. Si escribimos
7 asi \[ 1 = \int 3 \] el salto es
8 automatico. De hecho, LaTeX no
9 cambia de parrafo.

```

Podemos poner ecuaciones $1 = 2$ en línea, o tambien separadas con

$$1 = 2 + 4$$

No hace falta poner el salto de linea, se hace solo. Si escribimos asi

$$1 = \int 3$$

el salto es automatico. De hecho, LaTeX no cambia de parrafo.

A modo de aviso, en muchos sitios todavía se sigue usando dos dólares (\$\$) en lugar de los corchetes. No deberían usarse (es sintaxis antigua), y además puede dar algunos errores y generar text que no queda del todo bien.

Para usar matemáticas en un documento L^AT_EX, es recomendable incluir el paquete *amsmath*, que añade bastantes comandos y mejora el funcionamiento de las ecuaciones.

I.3.1. Subíndices y superíndices, fracciones y raíces

Para poner subíndices y superíndices, usaremos los caracteres `_` y `^` respectivamente. Por ejemplo:

```
1 \[ a_b + 3^2 \]
2 \[ a_{bj} + a_{bj} + 3^{2i} + 3^{2i} \]
```

$$a_b + 3^2$$

$$a_{bj} + a_{bj} + 3^{2i} + 3^{2i}$$

Hay que tener cuidado cuando querramos poner subíndices o superíndices con más de dos caracteres: en ese caso, hay que rodearlos con llaves porque si no LaTeX sólo pone el primer carácter.

Las fracciones se ponen con el comando `\frac{numerador}{denominador}`, o `\dfrac` si estamos en modo *inline* (con `$...$`) y queremos que la fracción no aparezca en tamaño pequeño.

Para poner raíces usaremos el comando `\sqrt`, que como argumento opcional acepta el orden de la raíz:

```
1 Las fracciones en el texto \frac{1}{2}
2 aparecen reducidas, cosa que
3 podemos corregir con \dfrac{3}{4}.
4
5 \[ \frac{3 + 2}{5} = 1 = \sqrt{1} = \sqrt[3]{1} \]
```

Las fracciones en el texto $\frac{1}{2}$ aparecen reducidas, cosa que podemos corregir con $\frac{3}{4}$.

$$\frac{3 + 2}{5} = 1 = \sqrt{1} = \sqrt[3]{1}$$

I.3.2. Matrices

Las matrices en LaTeX tienen una sintaxis parecida a las tablas (sección 1.2.3.3): separamos las celdas por `&` y las filas por `\\`. La diferencia está en que usamos el entorno *matrix* (o sus variantes *pmatrix* o *vmatrix*):

```
1 \[
2 \begin{matrix}
3 1 & 2 \\
4 3 & 4
5 \end{matrix} =
6 \begin{pmatrix}
7 1 & 2 \\
8 3 & 4
9 \end{pmatrix} =
10 \begin{vmatrix}
11 1 & 2 \\
12 3 & 4
\end{vmatrix}
```

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}$$

I.3.3. Delimitadores

En ocasiones querremos introducir delimitadores, como paréntesis o llaves, en las ecuaciones. Normalmente funcionan bien, salvo cuando queremos meter cosas más grandes dentro (por ejemplo, un símbolo de sumatorio) que se quedan limitadores demasiado pequeños. Para evitar eso usaremos los comandos `\left` y `\right` seguidos

de un carácter que L^AT_EX cambiará de tamaño para que se ajuste bien. Si por lo que sea queremos que uno de los lados no tenga ningún delimitador, podemos usar un punto. Veámoslo claro con un ejemplo:

```
1 \[ (1 + 2) = 3 \]
2 \[ (\frac{3}{4} + 4) = \left(\frac{3}{4} + 1\right) \]
3
4 % Podemos tener delimitadores distintos
5 % o incluso quitar uno de ellos con '.'
6 \[ \left(\sum \right) = \left(\int \right) \]
```

$$(1 + 2) = 3$$

$$\left(\frac{3}{4} + 4\right) = \left(\frac{3}{4} + 1\right)$$

$$\left(\sum\right) = \left(\int\right)$$

Siempre hay que acordarse de cerrar bien los delimitadores: si nos dejamos un `\left` sin su correspondiente `\right` (o viceversa) L^AT_EX nos dará un error de compilación un poco raro.

I.3.4. Símbolos, operadores y texto

Es obvio que, salvo unos pocos casos, necesitaremos símbolos complejos para nuestras ecuaciones. Prácticamente todos están definidos como comandos, incluyendo las letras griegas. Algunos, como los comandos de límite, integral o sumatorio, colocan los superíndices y subíndices correctamente cuando los ponemos. Veamos ejemplos de esos símbolos ([aquí hay una lista más completa](#)):

```
1 \begin{gather*}
2 \forall x \in J, \exists y \in J \implies z
   \pm \infty \iff z = 1 \\
3 \lim_{n \rightarrow 0} \delta = \int_a^b 3 \\
4 \sum_{i=1}^n A \leq \prod_{j=1}^K m \\
   \emptyset \subset A_i = \bigcup B_j \subseteq \partial \Phi
5 \emptyset \subset A_i = \bigcup B_j \subseteq \partial \Phi
6 \end{gather*}
```

$$\forall x \exists y \in J \implies z \pm \infty \iff z = 1$$

$$\lim_{n \rightarrow 0} \delta = \int_a^b 3$$

$$\sum_{i=1}^n A \leq \prod_{j=1}^K m$$

$$\emptyset \subset \bigcup A_i = \bigcap B_j \subseteq \partial \Phi$$

L^AT_EX también tiene definidos comandos para los operadores habituales, como funciones trigonométricas o logaritmos, para que aparezcan correctamente. Si no usamos esos comandos, nos quedarán ecuaciones bastante feas.

```
1 \[ \cos a = \log b + \arccos 3 \]
2 \[ \cos a = \log b + \arccos 3 \]
```

$$\cos a = \log b + \arccos 3$$

$$\cos a = \log b + \arccos 3$$

Si por otra parte queremos poner texto en la ecuación, tendremos que usar el comando `\text` para que quede bien

```
1 \[ f \in L \implies f \text{ es inyectiva} \]
2 \[ f \in L \implies f \text{ es inyectiva} \]
```

$$f \in L \implies f \text{ es inyectiva}$$

$$f \in L \implies f \text{ es inyectiva}$$

Por último, a veces querremos forzar un espaciado concreto. Para ello podemos usar los comandos correspondientes: usar alguno de los comandos de espaciado `\ , \; \quad \qquad`.

```
1 \[ \forall f \in L, L \implies f \text{ es inyectiva} \]
```

$$\forall f \in L \implies f \text{ es inyectiva}$$

I.3.5. Caracteres en otras fuentes

Muchas veces en matemáticas se usan caracteres con otras fuentes, como las caligráficas, góticas o dobles. El paquete *amsfonts* proporciona los comandos necesarios:

```
1 \[ \mathbb{RCNZ}; \mathrm{upright}; \\ \mathfrak{ABCG}; \mathcal{CPE} \]
```

$$\mathbb{RCNZ} \text{ upright } \mathfrak{ABCG} \mathcal{CPE}$$

I.3.6. Entornos matemáticos

El paquete *amsmath* viene con un buen número de entornos matemáticos para organizar ecuaciones. Hay muchos, y están bien documentados. Sin embargo, vamos a describir los más interesantes:

- *equation* Es igual que poner `\[. . . \]`, con la diferencia de que nos pone un número de ecuación para que podamos referenciarla más tarde.
- *multline* Nos permite romper las ecuaciones en varias líneas cuando son muy largas, usando el comando `\\`, y deja la alineación correcta.
- *align* Nos permite alinear las ecuaciones, usando el carácter `&` como separador.
- *gather* Agrupa varias ecuaciones, centrándolas todas.

Todos estos entornos cuentan con versiones “*” (por ejemplo, *multline**) que hacen lo mismo pero sin incluir números de ecuación.

```

1 \begin{equation} \label{eq:Ecuacion}
2 3 + 4 = 123
3 \end{equation}
4
5 En la ecuacion \eqref{eq:Ecuacion}...
6
7 \begin{multline*}
8 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + \\\
9 + 10 + 11 + 12 + 13 + \sum_{i=14}^{\infty} i
10 \end{multline*}
11
12 \begin{align}
13 f(x) + \lambda &= g(x) + a \\\
14 f'(x) &= g'(x) + \frac{a}{x} = \\\
15 &= 4g'(x)^2
16 \end{align}
17
18 \begin{gather*}
19 f(x) = g(x) + a \\\
20 k(x) = g'(x) + \frac{\partial a}{\partial x}
21 \end{gather*}

```

$$3 + 4 = 123 \quad (\text{I.2})$$

En la ecuacion (I.2)...

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + \\ + 10 + 11 + 12 + 13 + \sum_{i=14}^{\infty} i$$

$$f(x) + \lambda = g(x) + a \quad (\text{I.3})$$

$$f'(x) = g'(x) + \frac{a}{x} = \quad (\text{I.4})$$

$$= 4g'(x)^2 \quad (\text{I.5})$$

$$f(x) = g(x) + a$$

$$k(x) = g'(x) + \frac{\partial a}{\partial x}$$

I.4. Preguntas frecuentes

Enhorabuena: ya deberías saber LaTeX básico. Ahora, como buen manual, incluimos una sección de preguntas frecuentes.

Mi documento no compila y LaTeX no me dice por qué: ¿qué hago? Suele ser obligatorio acordarse de los familiares de los desarrolladores de LaTeX por no poner errores más descriptivos. Una vez hecho eso, lo más útil suele ser mirar en qué línea ha fallado y buscar sospechosos habituales: comandos de matemáticas que están fuera del modo matemáticas, una barra baja que no está metida en el modo matemático...

A veces ocurre que el compilador no muestra el error en la línea que es, sino mucho después cuando se le acumulan tantos fallos que no es capaz de seguir. En ese caso suele ser útil revisar el registro de errores para ver qué más hay: a veces hay cosas útiles.

Otra opción es que los argumentos de la línea de comandos no sean los correctos. En concreto, uno que suele dar problemas es `-shell-escape`, que a grandes rasgos permite que el documento a compilar haga llamadas a otros comandos del sistema. Si esa opción no está en la línea de comandos del compilador, puede dar problemas y fallar con errores raros.

En cualquier caso, LaTeX no suele ser muy descriptivo con los mensajes de error, así que lo más recomendable es compilar el documento frecuentemente para detectar los fallos antes y no tener que navegar por un montón de líneas que acabemos de cambiar.

Los operadores matemáticos están muy juntos El espaciado de algunos símbolos matemáticos no acaba de cuadrar en algunos casos, como por ejemplo $\forall a \in E \exists y(a) \in B$. En este caso, es recomendable usar los operadores de espaciado `\ , \ ; \quad \qquad` (ver sección I.3.4). Normalmente `\ ,` es lo que mejor queda, pero

esto va a gusto del consumidor.

Los subíndices me salen pequeños cuando uso el modo *inline* Cuando se usan matemáticas en modo *inline* algunos operadores cambian el lugar de los subíndices y superíndices para que no ocupen mucho, quedando algo así: $\sum_{i=1}^k$. Si afecta, lo más probable es que quede mejor pasando la ecuación a una línea separada usando `\[...]`. Sin embargo, se puede añadir `\displaystyle` al principio de la ecuación para que ponga los subíndices más claros:

```
1 En línea \sum_{i=1}^k y algo mejor
2 con \displaystyle\sum_{i=1}^k, aunque
3 quizás sea mejor ponerlo separado con
4 \[ \sum_{i=1}^k ]
```

En línea $\sum_{i=1}^k$ y algo mejor con $\sum_{i=1}^k$, aunque quizás sea mejor ponerlo separado con
$$\sum_{i=1}^k$$

Capítulo II

L^AT_EX avanzado

Una vez que ya nos manejamos con las cosas básicas de L^AT_EX, vamos a pasar a cosas más avanzadas, como las etiquetas, índices, marcos y Tikz. También veremos parte de desarrollo de paquetes y clases de L^AT_EX: cómo crear comandos y entornos y modificarlos para que hagan lo que nosotros queremos. Para eso, antes veremos cómo funciona exactamente L^AT_EX, que nos ayudará a la hora de entender por qué tenemos que hacer ciertas cosas, como compilar dos veces, o qué es lo que puede estar fallando cuando algo no funciona.

II.1. ¿Cómo funciona L^AT_EX?

El capítulo anterior pretendía ser un tutorial básico de L^AT_EX, para que podamos ponernos a escribir rápidamente. Pero para avanzar, creo que es importante saber cómo funciona este sistema, más que nada para entender de dónde salen los errores y saber exactamente qué es lo que estamos haciendo. Además, como curiosidad informática tampoco viene nada mal.

Para ser precisos, más que de L^AT_EX deberíamos hablar de T_EX, que es el “núcleo” escrito por Donald Knuth. Este núcleo es un compilador que, a grandes rasgos, tiene dos fases: la de expansión y la de compilación.

Cuando compilamos un documento para generar un PDF, lo que va haciendo T_EX es ir leyéndolo carácter a carácter, e ir traduciéndolo. Los caracteres normales se pasan sin modificar, y los comandos (los que empiezan por `\`, como `\textbf`) se expanden.

La expansión de un comando no es más que sustituirlo por su definición sustituyendo los argumentos. Por ejemplo, yo puedo definir el comando `\vector`, que recibe un argumento (denotado por `#1`) y que se expande como `$\mathbf{#1}$`. Así, cuando T_EX encuentre `\vector{a}` lo sustituirá por `\mathbf{a}`. Este proceso se va realizando de forma recursiva: cuando el compilador expande un comando y se encuentra que la expansión ha metido más comandos, los expande a ellos también. Todo esto se hace en orden de izquierda a derecha.

Así, comando a comando, T_EX va expandiendo el documento hasta llegar a la forma más básica, que son los comandos primitivos de T_EX, que simplemente dicen cosas tan de bajo nivel “añade este carácter a la línea” o “ahora usa esta fuente”. Con esos comandos, T_EX va decidiendo cómo va a ser la estructura del PDF que va a generar (los saltos de línea o de página, o la colocación de las imágenes, por ejemplo).

Finalmente, esa información se pasa a la “segunda etapa” de T_EX, que transforma

esos comandos primitivos en un formato estándar para un documento que nosotros podamos visualizar: normalmente es PDF, pero también pueden ser otros como DVI.

La parte específica de L^AT_EX consiste en comandos adicionales que facilitan la vida al usuario, pero a grandes rasgos el proceso de compilación es el mismo.

En la práctica, hay algunas cosas algo más complicadas, pero básicamente así funciona L^AT_EX. Lo primero que leerá del documento será el `\documentclass`, que le dirá qué archivo base cargar: ese archivo cargará los comandos de L^AT_EX básicos y preparará el documento (tipo de fuente, márgenes, papel, etc). Después, las sentencias `\usepackage` cargarán otros documentos con más macros y comandos, y en cuanto empiece el documento con `\begin{document}` el compilador empezará a procesar todos esos caracteres y comandos escritos, expandiendo estos últimos hasta llegar a las instrucciones primitivas para mostrar tu PDF.

¿Esto es importante? Algo sí, más que nada porque nos explica de dónde salen dos tipos problemas muy habituales. El primero son los relacionados con la expansión de comandos: L^AT_EX no es como un lenguaje de programación donde primero se evalúan los argumentos y luego se le pasan a la función. Aquí es al revés: primero se desarrolla la función (el comando) y luego se procesan los argumentos. Y si en algún momento algo no cuadra (cosa probable) la compilación falla.

La otra cuestión es algo que no hemos dicho explícitamente, pero que se puede ver: T_EX es un sistema de una sola pasada. Lee el fichero y va procesando según le llega, pero en ningún momento mira hacia atrás ni hacia delante¹. El principal inconveniente es que a veces tendremos que recompilar el documento para que pille todos los datos, como las referencias. Las referencias las escribe L^AT_EX en un documento aparte cuando se las encuentra, así que si ponemos una referencia antes de declarar la etiqueta, no va a saber resolverlo. En la segunda pasada ya se encontrará esa etiqueta declarada en el archivo aparte así que ya podrá rellenar el comando `\ref` correctamente.

Para más información sobre cómo funciona L^AT_EX, es recomendable echarle un ojo al libro *T_EX for the Impatient* o *The T_EXBook*.

II.2. L^AT_EX para usuarios avanzados

A lo largo de esta sección vamos a ver algunos aspectos de L^AT_EX que o bien nos dan más opciones para crear mejores documentos, nos facilitan la vida a la hora de escribir documentos o nos permiten personalizar ciertas partes del documento. Y todo esto usando comandos estándar y sin tener que desarrollar nada. Todavía.

La idea de la sección es ir de las partes más sencillas a las más difíciles, aunque esto siempre es subjetivo así que es recomendable echarle un ojo a todo por si acaso.

II.2.1. ¿Dónde están documentados los paquetes?

Aunque en el mundo de la informática no suele ser lo habitual, la mayoría de los paquetes de L^AT_EX que podamos usar están bien documentados, con documentación que es accesible desde tu ordenador. Lo más fácil es usar el comando *texdoc*: si desde la

¹Por ejemplo, redefinir un comando a mitad del documento sólo afecta a partir de ese punto, no antes.

terminal ejecutas `texdoc nombrepaquete` se abrirá un PDF con la documentación del paquete.

Si no, en Internet suelen estar las soluciones. Además de tener el repositorio central CTAN, también está [TeX - Stack Exchange](#) para resolver dudas y, por supuesto, Google.

II.2.2. Cambiando los márgenes del documento

L^AT_EX viene con unos márgenes determinados por defecto según la clase que usemos. Por suerte, se puede modificar fácilmente, simplemente con el paquete *geometry*. La sintaxis es muy sencilla, poniendo en el preámbulo (antes del `\begin{document}`) lo siguiente:

```
\usepackage[left=3cm, right=1cm, top=3cm, bottom=2cm]{geometry} % Márgenes
```

Como es fácil adivinar, *left*, *right*, *top*, *bottom* se refieren a los márgenes izquierdo, derecho, superior e inferior del documento. El paquete tiene varias opciones más, todas documentadas apropiadamente en su *texdoc* (ver sección II.2.1), pero en general son bastante menos habituales y no hace falta que las revisemos aquí.

II.2.3. Manejando documentos grandes con varios archivos

Una ventaja de L^AT_EX es que es muy fácil manejar documentos grandes, con muchas líneas. Si vemos que mantenerlo todo en un mismo archivo se hace incómodo, podemos separarlo en varios archivos distintos e importarlos en el documento principal con el comando `\input`.

```
\documentclass{article}
```

```
\begin{document}
```

Empezamos aquí con una introducción...

```
\input{Seccion1.tex}
```

```
\input{Seccion2.tex}
```

LaTeX meterá automáticamente el contenido de los archivos que le hemos dicho y compilará todo como un único documento.

```
\end{document}
```

El comando `\input` recibe como argumento el nombre del archivo a incluir. Si el archivo está en otro directorio (por ejemplo, en un subdirectorio en la misma carpeta) tendremos que poner la ruta relativa (p.e., `\input{tex/Seccion1.tex}`).

II.2.4. Cómo personalizar la tabla de contenidos y la numeración de secciones

La tabla de contenidos de L^AT_EX se genera automáticamente a partir de las secciones y subsecciones del documento. Aunque los valores por defecto suelen estar bien, a veces querremos cambiarlo. Por ejemplo, si queremos que aparezcan secciones,

subsecciones y subsubsecciones en la tabla de contenidos, tendremos que poner lo siguiente en el preámbulo

```
\setcounter{tocdepth}{3}
```

El 3 es la profundidad máxima de la tabla (sin contar capítulos). Si quisiésemos sólo secciones, podríamos poner un 1 en su lugar, por ejemplo.

LaTeX también nos permite cambiar cómo aparecen los contadores de sección, subsección y demás. La sintaxis en general es la siguiente:

% En general:

```
\renewcommand{\the[contador]}{\the[otrocontador]. \forma{[contador]}}
```

% Ejemplo: Secciones con numeros romanos

```
\renewcommand{\thesection}{\Roman{section}}
```

% Otro mas: Subsecciones con el numero de seccion de antes

% y contador en numeros.

```
\renewcommand{\thesubsection}{\thesection .\arabic{subsection}}
```

% En realidad podemos poner lo que queramos en el segundo

% argumento:

```
\renewcommand{\thesubsubsection}{Esta es la subseccion \alph{subsection}}
```

Podemos modificar los contadores que queramos: todos tienen el mismo nombre que el comando correspondiente (p.e., `\section` tiene como contador *section*). En cuanto a las formas de imprimir el número concreto, tenemos cinco posibilidades: `\arabic`, que nos saca números (1,2,3, ...); `\alph` o `\Alph` que nos sacará letras (a,b,c) en minúsculas o mayúsculas; y `\roman` o `\Roman` que nos sacarán números romanos (I, II, IV...) en minúsculas y mayúsculas respectivamente.

II.2.5. Referencias todavía más automáticas: fancyref

Hay un paquete adicional que mejora las características de las referencias de LaTeX que se llama *fancyref*². Este paquete introduce el comando `\fref`, que saca la referencia con el nombre que le corresponde y con el enlace. Además, *fancyref* tiene un formato “vario” que muestra la página en la que está lo que estemos referenciando.

```
1 Antes usabamos \ref{eq:SuperImportante},
2 pero tambien podemos hablar de la
3 \fref{eq:SuperImportante}, de la
4 \fref{sec:EstructuraDocumento} o
5 de la \fref[vario]{sec:Tablas}.
```

Antes usabamos I.1, pero tambien podemos hablar de la ecuación (I.1), de la sección I.1.2 o de la sección I.2.3.3 en la página 9.

fancyref depende de que las etiquetas tengan un cierto nombre, de la forma *prefijo:Identificador*, para poner el nombre correcto. Estos prefijos están en la tabla II.1.

Adicionalmente, hemos creado el paquete *fancysprefs* que traducen los prefijos de *fancyref* al español y añade algunos comandos para incluir el nombre de lo que estamos referenciando automáticamente. Este paquete está descrito en la sección III.3.2.

²Para poder usar *fancyref* en el documento, sólo hay que añadir `\usepackage{fancyref}` en el preámbulo del documento. Ver la sección I.1.2 para saber qué es exactamente el preámbulo.

Elemento	Prefijo
Capítulo	chap
Sección	sec
Ecuación	eq
Figura	fig
Tabla	tab
Ejercicio	ej
Proposición	prop
Lema	lem
Teorema	thm
Definición	def

Tabla II.1: Prefijos establecidos para que *fref* funcione correctamente.

II.2.6. Estilos de página: pies y cabeceras

Podemos personalizar los estilos de página como queramos usando el paquete *fancyhdr* y configurándolo en el preámbulo. Por ejemplo, este código pondría una cabecera y un pie de página con el número de página.

```
\pagestyle{fancy}
\fancyhf{}

\rhead{Título del documento}
\lhead{Sección \thesection}
\cfoot{Página \thepage}

\renewcommand{\headrulewidth}{2pt} % Anchura de la línea de la cabecera
\renewcommand{\footrulewidth}{1pt} % Anchura de la línea del pie
% En ambos casos, poniendo 0pt desactivamos la línea correspondiente.
```

Aunque sólo hemos usado tres comandos, *fancyhdr* provee varios comandos del tipo `\[rcl][head|foot]{Contenido}` que nos permiten personalizar lo que ponemos en la parte derecha, izquierda o central de la cabecera o pie de página, respectivamente. Aquí viene bien recordar los contadores que vimos sección II.2.4, porque podemos usar esos sin problemas para referencias la sección actual.

También nos permite, como se puede ver en el código, la anchura de las líneas que usamos para separar la cabecera y el pie de página.

Para personalizar las cosas un poco más, hay dos posibilidades interesantes: si ponemos el paquete *lastpage* podremos usar `\pageref{LastPage}` para imprimir el número de páginas del documento. Así, podríamos poner `\cfoot{\thepage de \pageref{LastPage}}` para tener un pie de página como el de este manual.

Si lo que queremos es poner el nombre de sección o de capítulo está algo más complicado. Los comandos `\leftmark` y `\rightmark` sacan el primer nivel (sección normalmente, o capítulos si los estamos usando) y el segundo respectivamente, pero cambiar el formato o usar cosas distintas no es trivial y hay que cambiar algunos comandos.

II.2.7. Dibujos y gráficas de funciones con código LaTeX: Tikz

A estas alturas, ya debería quedar claro que LaTeX permite hacer bastantes cosas. Pero por si acaso, vamos a hacer una introducción a Tikz, que es el sistema que tiene LaTeX para poder hacer dibujos directamente en el documento con comandos.

Tikz es un sistema impresionantemente completo. El manual tiene más de 1.000 páginas, para que nos hagamos una idea. También tienen una [introducción rápida, en inglés](#), con las cosas básicas y algunos ejemplos muy interesantes. Igualmente por Internet hay muchos recursos, como [cheat-sheets de referencia rápida](#) o una extensa [colección de ejemplos](#).

Precisamente por lo completo que es, aquí simplemente vamos a introducirlo con ejemplos, comentando que se puede hacer e invitando al lector interesado a leer más en Internet, en el manual o experimentando por sí mismo.

Para usar Tikz, primero hay que activarlo en el preámbulo con `\usepackage{tikz}`. Después, cuando queramos hacer un dibujo, tenemos que meter los comandos de Tikz en un entorno `tikzpicture`. Nota importante: los comandos de Tikz acaban *siempre* en punto y coma: si no se pone, el compilador se puede atragantar y no acabar.

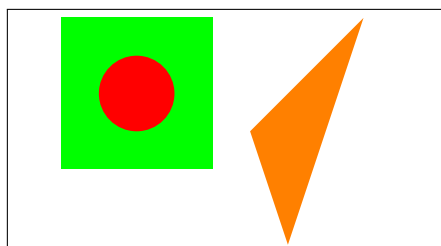
Vamos ahora con el ejemplo de comandos y de cosas que se pueden hacer. No pretende ser ejemplos completos, sino dar unas pinceladas para poder empezar a manejarse con Tikz. Para facilitar las cosas, ponemos junto con cada código los resultados que salen.

Aviso: Si el documento de LaTeX tiene configurado el idioma en español con babel (comando `\usepackage[spanish]{babel}`) y ponemos la flecha con `->`, va a reventar por todo lo alto. La solución es añadir las opciones `es-noquoting,es-noshorthands`, de tal forma que nos quede `\usepackage[spanish,es-noquoting,es-noshorthands]{babel}`.

```
1 \begin{tikzpicture}
2 % Sistema de coordenadas sencillo: (X,Y)
3 \draw (0,0) -- (1,0);
4 \draw[thick, red] (1,0) -- (1,1);
5
6 % '->' como opcin indica poner flecha
7 \draw[blue, ->] (1,1) -- (0,1);
8
9 % Podemos curvar lineas fcilmente
10 \draw[orange] (0,0) to[bend right] (1,0);
11
12 % Hay varios estilos de recta
13 \draw[dotted] (3,0) -- (4,0);
14 \draw[dashed] (3,1) -- (4,1);
15 \end{tikzpicture}
```



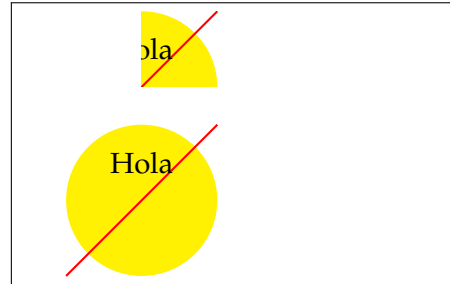
```
1 \begin{tikzpicture}
2 % Dibujar formas rellenas
3
4 % Con las dos esquinas ponemos el rectngulo
5 \fill [green] (0,0) rectangle (2,2);
6
7 % El segundo argumento es el radio del crculo.
8 \fill [red] (1,1) circle (0.5cm);
9
10 % Las formas pueden ser aleatorias
11 \fill [orange] (3,-1) -- (4,2) -- (2.5,0.5) --
    cycle;
12 \end{tikzpicture}
```



```

1 \begin{tikzpicture}
2 % Tambien podemos usar "clipping" para recortar.
   Eso si, hay que usarlo dentro de un entorno
   "scope" para evitar recortar todo el dibujo
   .
3 \begin{scope}
4 % Mostramos solo lo que este dentro de esta
   region que definimos con "clip".
5 \clip (0,0) rectangle (1,1);
6 \fill [yellow] (0,0) circle (1cm);
7
8 % Funciona con todo lo que pongamos, no solo
   fill
9 \draw[red, thick] (-1,-1) -- (1, 1);
10 \node at (0, 0.5) {Hola};
11 \end{scope}
12
13 % Vemos que pasa si ponemos lo mismo sin
   clipping.
14 \begin{scope}[yshift=-1.5cm] % Ventajas de scope
   : podemos mover una parte del dibujo con
   xshift/yshift. Tambien podriamos usar scale
   = 2 para doblarle el tamano, por ejemplo.
15 \fill [yellow] (0,0) circle (1cm);
16 \draw[red, thick] (-1,-1) -- (1, 1);
17 \node at (0, 0.5) {Hola};
18 \end{scope}
19 \end{tikzpicture}

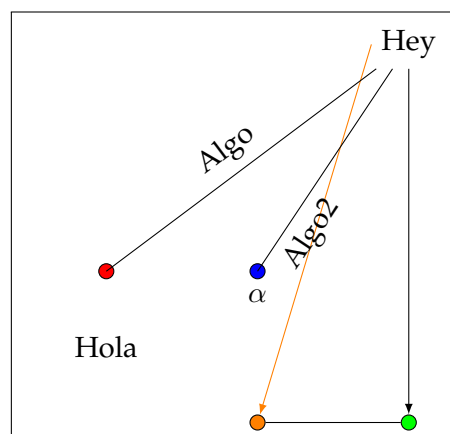
```



```

1 \begin{tikzpicture}
2 % Podemos poner nodos con texto o sin texto. Las
   llaves del final son necesarias siempre.
3 \node at (0, 1) {Hola};
4 \node[draw, circle, inner sep = 2pt, fill = red]
   at (0,2) {};
5
6 % Tambien se pueden poner etiquetas
7 \node[draw, circle, inner sep = 2pt, fill = blue
   , label = below:{$\alpha$}] at (2,2) {};
8
9 % Los nodos pueden ir nombrados para
   referenciarlos luego
10 \node[draw, circle, inner sep = 2pt, fill =
   orange] (A) at (2,0) {};
11 \node[draw, circle, inner sep = 2pt, fill =
   green] (B) at (4,0) {};
12 \draw (A) -- (B);
13
14 % Podemos hacerlo igual con los nodos con texto
15 \node (C) at (4, 5) {Hey};
16 \draw[->] (C) -- (B);
17
18 % Podemos decidir de que parte del nodo (south,
   west, north, east) sale la flecha
19 \draw[orange, ->] (C.west) -- (A);
20
21 % Y tambien podemos poner nodos en lineas
22 \draw (0,2) -- node[midway, above, sloped] {Algo
   } (C);
23 \draw (2,2) -- node[near start, below, sloped] {
   Algo2} (C);
24 \end{tikzpicture}

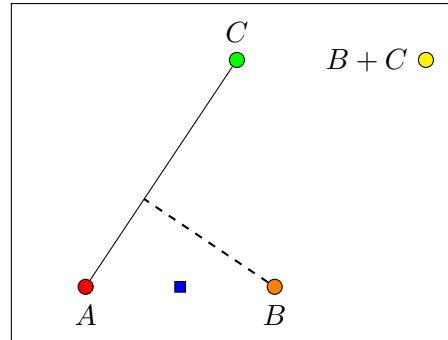
```




```

1 \usetikzlibrary{calc} % Para hacer calculos de
  puntos
2
3 \begin{tikzpicture}
4 % Dibujamos tres vertices de un triangulo
5 \node[draw, circle, inner sep = 2pt, fill = red,
  label = below:{$A$}] (A) at (0,0) {};
6 \node[draw, circle, inner sep = 2pt, fill =
  orange, label = below:{$B$}] (B) at (2.5,0)
  {};
7 \node[draw, circle, inner sep = 2pt, fill =
  green, label = above:{$C$}] (C) at (2,3) {};
8
9 % Ahora podemos hacer calculos y dibujar nodos
  donde queramos.
10 % Por ejemplo, un nodo a medio camino entre A y
  B
11 \node[draw, inner sep = 2pt, fill = blue] at ($(
  A)!0.5!(B)$) {};
12
13 % O sacar la recta perpendicular a AC que pasa
  por B
14 \draw (A) -- (C);
15 \draw[thick, dashed] (B) -- ($(A)!(B)!(C)$);
16
17 % Podemos sumar coordenadas
18 \node[draw, circle, inner sep = 2pt, fill =
  yellow, label = left:{$B + C$}] at ($(B) + (
  C)$) {};
19 \end{tikzpicture}

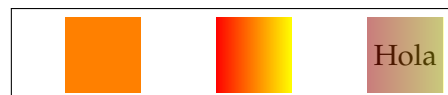
```



```

1 \begin{tikzpicture}
2 % Con colores tambien podemos hacer cosas chulas
  . Podemos combinarlos para sacar cosas
  distintas:
3 \fill [red!50!yellow] (0,0) rectangle (1,1);
4
5 % O hacer degradados
6 \fill [left color = red, right color = yellow]
  (2,0) rectangle (3,1);
7
8 % Y transparencias
9 \node at (4.5, 0.5) {Hola};
10 \fill [left color = red, right color = yellow,
  opacity = 0.3] (4,0) rectangle (5,1);
11 \end{tikzpicture}

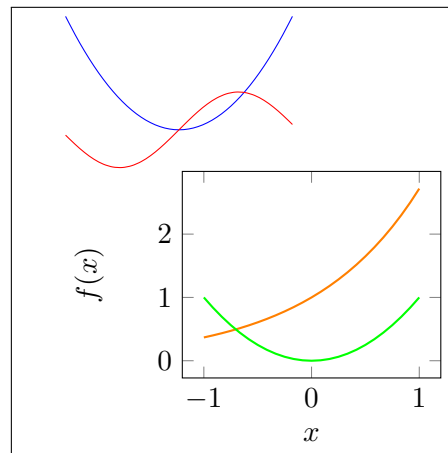
```



```

1 \begin{tikzpicture}
2 % Tambien podemos hacer funciones
3 \draw[scale=0.5,domain=-3:3,smooth,variable=\x,
  blue] plot ({\x},{\x*\x / 3});
4 \draw[scale=0.5,domain=-3:3,smooth,variable=\x,
  red] plot ({\x},{sin(\x r)});
5 % Nota: los senos y cosenos tikz los trata con
  grados. Hay que poner la r para que
  convierta a radianes.
6 \end{tikzpicture}
7
8 \begin{tikzpicture}
9 % Tambien se puede usar el entorno axis, que es
  mas facil de usar si solo queremos hacer
  graficas.
10 % Para activarlo, hay que poner \usepackage{
  pgfplots} en el preambulo.
11 \begin{axis}[width=5cm, domain = -1:1, xlabel =
  $x$, ylabel = {$f(x)$}]
12 \addplot[color=orange, thick]{exp(x)};
13 \addplot[color=green, thick]{x^2};
14 \end{axis}
15 \end{tikzpicture}

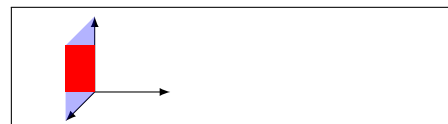
```



```

1 \begin{tikzpicture}
2 % Tikz tambien se maneja con coordenadas 3D
3 \draw[>-] (0,0,0) -- (1,0,0);
4 \draw[>-] (0,0,0) -- (0,1,0);
5 \draw[>-] (0,0,0) -- (0,0,1);
6
7 % Aunque no del todo bien. Esto funciona
8 \fill [blue, opacity = 0.3] (0,0,0) -- (0,1,0) --
  (0,1,1) -- (0,0,1) -- cycle;
9
10 % Pero esto no como queriamos
11 \fill [red] (0,0,0) rectangle (0,1,1);
12
13 % Para cosas en 3D mas avanzadas, ver
14 % tikz-3dplot
15 \end{tikzpicture}

```



II.2.8. Un compilador más completo: latexmk

Al principio de este capítulo veíamos cómo funcionaba L^AT_EX (sección II.1), y que uno de los problemas es que a veces hay que hacer varias pasadas para que funcione todo bien, lo cual puede ser un poco pesado. Por suerte, hay una solución llamada *latexmk*, que es un compilador³ que se encarga de hacer todo esto de manera automática. Simplemente ejecutando `latexmk -pdf -silent documento.tex` compilará todo, haciendo tantas pasadas como haga falta.

La segunda ventaja de *latexmk* es que nos permite compilar el documento de forma continua: cada vez que lo guardamos, *latexmk* recoge los cambios y recompila para que nuestro lector de PDF pueda actualizar el documento⁴. Para ello sólo tenemos que añadir el parámetro *-pvc*: `latexmk -pdf -silent -pvc documento.tex`.

³En realidad es un script en Perl que llama a pdflatex y demás compiladores según convenga, pero para el caso es un compilador.

⁴En Linux, Evince y Okular recargan el documento automáticamente. En OS X, Skim funciona muy bien.

latexmk tiene varias opciones más (ver su página de manual) y merece la pena probarlo y usarlo para quitarnos de problemas. Suele estar disponible como un paquete normal en las distribuciones Linux (`sudo apt-get install latexmk` funciona, por ejemplo) y también en MacPorts y Brew para OS X.

II.2.9. Beamer: presentaciones con L^AT_EX (o cómo dije adiós a Powerpoint)

Entre todas las cosas que permite hacer L^AT_EX una de ellas es presentaciones con *beamer*. Es una clase de documento que nos permitirá crear transparencias con toda la potencia de L^AT_EX y olvidándonos de Powerpoint. Se escribe todo exactamente igual que en un documento normal, salvo porque tenemos que agrupar el contenido en transparencias con `\begin{frame}... \end{frame}`. No vamos a entrar mucho en detalle: simplemente pretendo dar una idea de que esto existe y es útil. Para una guía completa, [la documentación de Beamer está muy bien](#). También recomendaría echarle un ojo a los temas de Beamer para cambiar el estilo. [Aquí hay un directorio de los que vienen por defecto](#), aunque yo personalmente prefiero *mntheme*, que es bastante más ligero y con una tipografía bastante mejor.

```
\documentclass{beamer}

\usetheme{default} % O cualquier otro

\title{Adiós, Powerpoint}
\date{\today}
\author{Yo}

\begin{document}
  \maketitle

  \section{Sección}

  \begin{frame}{Título}
    \framesubtitle{Subtítulo}

    Hello, world!
  \end{frame}
\end{document}
```

II.3. Ampliando L^AT_EX: Desarrollo de comandos, entornos, paquetes y clases

Una de las ventajas de L^AT_EX es que nos permite “programar” nuestros documentos, creando nuevos comandos y entornos y así automatizar todo lo posible nuestra escritura. Lo malo es que la sintaxis no es la más cómoda del mundo, y tiene bastantes cosas algo impredecibles. En cualquier caso, aquí vamos a ver una introducción sencilla a la programación de comandos y entornos en L^AT_EX.

II.3.1. Comandos y entornos

A estas alturas ya deberíamos conocer más o menos cómo va L^AT_EX (y si no, léase el comienzo de este capítulo): nosotros escribimos comandos y el procesador los va expandiendo con los argumentos que le pasamos. Sólo nos falta saber cómo definir un comando, y eso es bastante sencillo.

```
% Definimos un nuevo comando
\newcommand{\micomando}{Contenido del comando}

% Si ahora escribimos esto:
\micomando
% En el documento aparecerá "Contenido del comando"

% Los comandos pueden recibir argumentos. Para ello, usamos la sintaxis
% \newcommand{comando}[número de argumentos]{definición}
% En la definición, los #1, #2, ..., #n se sustituirán por el argumento
% en la posición correspondiente.
\newcommand{\yosoy}[1]{Hola, soy #1}

\yosoy{Guille} % -> "Hola, soy Guille"

% Los comandos pueden aceptar un único argumento opcional. Para ello,
% después del número de argumentos ponemos el valor que tendrá por defecto
% el argumento. El argumento opcional siempre es el primero.
\newcommand{\opcional}[2][ ]{Primero #1 y segundo #2}

\opcional{dos} % -> Primero y segundo dos
\opcional[uno]{dos} % -> Primero uno y segundo dos

% El argumento opcional puede tener un valor por defecto
\newcommand{\opcional}[2][tres]{Primero #1 y segundo #2}
\opcional{dos} % -> Primero tres y segundo dos
```

De una forma bastante similar podemos definir los entornos, que son los que empiezan con `\begin{...}` y acaban con `\end{...}`.

```
% Sintaxis
\newenvironment{test}{esto se sustituye en el \begin{}}{esto se sustituye en el \end{}}

% Los argumentos siguen la misma sintaxis que para \newcommand.

% Ejemplo: un entorno con argumento opcional
\newenvironment{operacion}[1][ ]{\textsc{Operación #1}: }{\hline}

% Si ponemos esto
\begin{operacion}[prueba]
\[ 32 + 4 = 8\]
\end{operacion}

% Se sustituye después por:
```

```
\textsc{Operación prueba}: % La parte del \begin
\[ 32 + 4 = 8 \]
\hline % La parte del \end
```

Si por lo que sea el comando o entorno que queremos definir ya está definido, simplemente tendremos que usar `renewcommand` o `renewenvironment`, con la misma sintaxis que antes.

II.3.1.1. Manejo avanzado de argumentos

Como se puede ver, la definición de nuevos comandos es bastante sencilla. Lo único malo es que la gestión de los argumentos es un poco tirando a mala: en cuanto queramos tener varios argumentos opcionales o tengamos muchos parámetros, la implementación de LaTeX se nos va a quedar corta.

Para el primer problema, el paquete *xargs* da una solución: variantes `newcommandx` y `newenvironmentx`, que permiten definir múltiples argumentos opcionales. La sintaxis es muy sencilla y similar a la que acabamos de ver

```
% Sintaxis
\newcommandx{\comando}[nargs][1=valor por defecto arg1, ...]{definicion}

% Ejemplo:
\newcommandx{\convs}[3][1=,2=n,3=\infty]{\xrightarrow[#2\to #3]{#1}}

\convs % -> \xrightarrow{n \to \infty}{}
\convs[c.s.] % -> \xrightarrow{n \to \infty}{c.s.}
\convs[c.s.][x] % -> \xrightarrow{x \to \infty}{c.s.}
\convs[c.s.][x][0] % -> \xrightarrow{x \to 0}{c.s.}
```

El problema de tener muchos argumentos ya no se resuelve tan fácilmente. Hay muchos paquetes para poner argumentos en la forma *clave = valor*. A mí personalmente me gusta *pgfkeys*, así que pondré un ejemplo sencillo aquí:

```
% Primero se definen los argumentos
\pgfkeys{
  % Antes de nada, se agrupan en una "familia" las claves
  % y le decimos a pgf que entramos a definir esa familia
  /testkeys/.is family,
  /testkeys,

  % name será una clave, cuyo valor se guardará
  % en el comando \aname
  name/.store in = \aname,
  name/.value required,

  % Con .code podemos hacer que se ejecute un código
  % cuando nos pasan el argumento codearg. #1 se sustituye
  % por el valor correspondiente
  counter/.code = {\setcounter{test}{#1}},
```

```

% Con .is choice podemos definir una serie de argumentos permitidos
enum/.is choice,
enum/true/.code = {\def\enumval{true}},
enum/false/.code = {\def\enumval{false}}

% Podemos definir un estilo "default" con valores por defecto
default/.style =
  { name={empty}, enum=true, counter = 0}
}

% Ahora definimos el comando con un argumento opcional
\newcommand{\testcommand}[1][]{
  % Ejecutamos las claves opcionales, pasando primero "default"
  % para que, si #1 está vacío, se ejecuten las opciones por defecto
  \pgfkeys{/testkeys, default, #1}

  % Ahora podemos usar los comandos que hemos definido antes:
  El nombre es \aname, enum es \enumval y parece ser que tenemos
  un contador que vale \arabic{test}.
}

% Ejemplo:
\testcommand[name = test, counter = 2, enum = false]
  % -> El nombre es test, enum es false y parece ser que tenemos
  % un contador que vale 2

\testcommand
  % -> El nombre es empty, enum es true y parece ser que tenemos
  % un contador que vale 0

```

II.3.1.2. Comandos de bajo nivel T_EX y expansión de argumentos

Justo en el ejemplo anterior, he puesto un comando que no hemos comentado: `\def`. Lo he puesto más que nada como excusa para escribir esta sección. Ya voy avisando que nos estamos adentrando en cosas algo más técnicas de L^AT_EX/ T_EX y yo no tengo del todo lo claro lo que escribo aquí.

`\def` es simplemente la forma de definir comandos en T_EX. Por decirlo de alguna forma, es un manejo a más bajo nivel que L^AT_EX. En general, es recomendable usar `\newcommand` porque es más robusto, hace algunas comprobaciones necesarias (como por ejemplo, que no estés redefiniendo un comando que ya existe) y es más fácil de usar. La ventaja que tiene `\def` es que es más corto y para definir comandos sencillos, como en el ejemplo de arriba un `\def\enumval{true}` es más cómodo.

Junto con `\def`, me gustaría comentar el comando `\let`, que también es bastante útil: nos permite darle un nombre alternativo a un comando. Digo que es especialmente útil porque con él podremos poner “capas” por encima de otros comandos ya existentes. Por ejemplo, supongamos que queremos redefinir el comando `\section` para que, además de poner la sección, nos meta el título en el glosario con `\index`. Tenemos que redefinir `\section` pero no podemos eliminarlo del todo. Entonces haríamos algo así:

```

\let\oldsection\section % \oldsection tiene ahora el código de \section

% Así que ya podemos redefinir \section
\renewcommand{\section}[1]{
    \index{#1} % Hacemos lo que queremos nosotros antes
    \oldsection{#1} % Y ahora llamamos al comando \section de antes
}

```

Otra primitiva interesante de T_EX son las variables booleanas. Lo explico mejor con un ejemplo. Supongamos que queremos saber cuándo estamos en el apéndice del documento para que un comando cambie la salida. Entonces podríamos hacer algo así:

```

% En el preámbulo
\newif\if@inappendix
\@inappendixfalse % Por defecto estamos fuera del apéndice

\let\oldappendix\appendix % Nos guardamos \appendix
\renewcommand{\appendix}{ % Y lo redefinimos
    \@inappendixtrue % Cambiamos la variable a true
    \oldappendix % Y llamamos al comando \appendix viejo
}

% Ahora definimos el comando que sabe cuándo está en el apéndice
\newcommand{\mycommand}{
    \if@inappendix
        Estamos en el apéndice
    \else
        No estamos en el apéndice
    \fi
}

```

Y ya que estamos, voy a mencionar un tema peliagudo, que es el de expansión de argumentos y comandos. Como decíamos al principio de este capítulo, L^AT_EX funciona expandiendo argumentos y sustituyendo cosas: primero expande el comando y luego sus argumentos, así todo en orden. A veces, eso puede llevarnos a fallos, y queremos que se expandan primero los argumentos y después el comando, o también cambiar el orden de expansión de formas un poco extrañas.

Aquí es donde entran `\expandafter`, `\noexpand` y amigos similares, que controlan si el siguiente comando se expande o no. [Esta respuesta de TeX.SX](#) explica más o menos bien cómo funcionan. Lo menciono aquí porque es importante saber que estas cosas existen, aunque personalmente todavía no he pasado del momento de programación aleatoria pegando `\expandafter` y `\noexpand` hasta que el comando funciona.

II.3.2. Paquetes y clases

Para poder reutilizar el código fácilmente, L^AT_EX permite crear paquetes y clases de documentos. A grandes rasgos, no son más que archivos con comandos que se pegan “a pelo” en el documento en el que se incluyen. En ambos casos, eso sí, empiezan con una cabecera especial:

```
% Para clases (archivos .cls)
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{nombreclase}[YYYY/MM/DD Descripción]

% Para paquetes (archivos .sty)
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{nombrepaquete}[YYYY/MM/DD Descripción]
```

Cuando en un documento se incluye una clase con `\documentclass` o un paquete con `\usepackage`, el compilador de L^AT_EX busca los correspondientes archivos `.cls` y `.sty` en los directorios de la instalación (a estos directorios mueve el script *install* que tenemos en el repositorio las clases y paquetes que hemos creado), y también en el directorio donde está el documento `.tex` que estemos compilando.

Para más detalles sobre cómo escribir paquetes y clases de documentos, es muy recomendable leerse [L^AT_EX 2_ε for class and package writers](#).

II.3.3. Paquetes auxiliares para el desarrollo

Ya para acabar con este tema, pongo aquí una pequeña referencia de paquetes que son bastante útiles a la hora de desarrollar comandos. Recuerdo que se puede ver la documentación de cada uno de ellos en Internet o, desde la consola de comandos de tu sistema, con `texdoc nombrepaquete`.

- *etoolbox*: Varias herramientas que vienen bien para el desarrollo. Especialmente útiles los *hooks* o ganchos para poner código al final o al principio del documento y otros entornos.
- *ifthen*: Como su nombre indica, permite escribir fácilmente sentencias *if-then-else*.
- *mfirstuc*: Da un comando para poner el primer carácter de una palabra en mayúsculas.
- *xspace*: Provee el comando `\xspace`, para poner espaciado correcto detrás de comandos matemáticos, evitando poner espacios de más o de menos cuando definimos algo.
- *xstring*: Permite trabajar con cadenas, manipulándolas, comparándolas, etc.

Capítulo III

Manual paquetes extendidos

III.1. Introducción: ¿qué podemos hacer con los paquetes extendidos?

Mientras escribimos los apuntes, hay muchas cosas que repetimos, que tecleamos muchas veces o que nos gustaría hacer más rápido o de forma unificada. Al final hemos creado varios paquetes que nos permiten tener un formato unificado y con varios comandos para escribir más rápido.

El paquete principal es el *exmath.sty* (sección III.3.1), que contiene una gran cantidad de comandos (la lista está en el apéndice A) que permiten escribir cosas repetidas fácilmente. Por ejemplo, podemos escribir una aplicación sólo con un comando ($\text{\app1{f}{A}{B}}$) o usar atajos como \cvv en lugar de $\text{\vec{k}}_{\alpha}$ para sacar k_{α} . También hay algunos entornos que hacen cosas solitas, como por ejemplo los de teoremas o definiciones, que añaden notas al margen y entradas en el índice.

También tenemos dos clases de documentos, *apuntes* (sección III.3.6) y *ejercicios* (sección III.3.7), que se pueden poner en el *documentclass* en lugar de *article*, y que dan un formato más o menos estándar al documento y cargan varios paquetes habituales, como las cadenas en español, marcos, colores y, por supuesto, el *exmath.sty*.

Un paquete muy útil es *MathUnicode.sty* (sección III.3.3), que permite introducir letras griegas y otros caracteres no estándar desde el teclado al documento, de tal forma que podamos escribir α y no \alpha cada vez que queramos poner ese carácter (y muchos otros).

Por último, hay algunos paquetes auxiliares, como *fancysprefs* para referencias mejoradas (sección III.3.2), *fastbuild* para mejorar velocidad de compilación en documentos pesados (sección III.3.5) o *tikztools* con algunos comandos adicionales para Tikz (III.3.4).

III.2. Estructura de los paquetes extendidos

Todos los paquetes se encuentran en el directorio *Cosas guays LaTeX* (sic). Es importante que estén en este directorio, ya que es en el que busca el script de instalación de paquetes (archivo *install* en la raíz del repositorio) para moverlos al sistema.

En ese directorio están los paquetes (archivos *.sty*) que hemos desarrollado nosotros, junto con algunos de terceros que a veces desaparecen en algunas instalaciones de L^AT_EX.

También están las clases *apuntes.cls*, *ejercicios.cls* y *custombase.cls*: esta última es la clase básica que comparten *apuntes* y *ejercicios*.

También en *Cosas guays LaTeX* está este manual, en la carpeta *Manual*. Ahí están las fuentes *.tex* de este documento y además el archivo *commandsampler.py* para sacar unos ejemplos a partir de un archivo *.sty*: ver la sección III.3.1.5 para una descripción algo más detallada.

Por último, dentro del directorio hay varios archivos adicionales:

- *AjustesTeXStudio.txspfile*: un archivo con ajustes para TeXStudio que vienen bastante bien, incluyendo archivos de completado por defecto.
- *cwlcreator.py*: un script en Python que genera archivos CWL a partir de un paquete *.sty*. Estos archivos se pueden incluir luego en Sublime o en TeXStudio para tener autocompletado de los paquetes que hemos desarrollado.
- Archivos **.cwl*: archivos de autocompletado para Sublime o TeXStudio.
- *ConfiguracionTeclado*: Configuración para el teclado con caracteres especiales (ver sección III.3.3).

III.3. Documentación de los paquetes

III.3.1. *exmath.sty*: Comandos y entornos adicionales de matemáticas

III.3.1.1. Entornos para teoremas y similares

Los teoremas del paquete heredan de los teoremas de *amsthm*, pero además añaden un índice automático de teoremas que se puede imprimir en el documento con `\printtheorems`. También se cargan automáticamente en el glosario de términos. El uso normal es el siguiente

```
1 \begin{theorem}[Teorema de las gallinas
   cluecas]
2 Teorema.
3 \end{theorem}
```

Teorema III.1 (Teorema de las gallinas cluecas). *Teorema.*

El título se carga automáticamente como una entrada para el índice. Si, por lo que sea, queremos separar los términos (por ejemplo, para que en este caso aparezca primero *Teorema* y luego, debajo, *de las gallinas cluecas*, ved la última página), no podemos usar la exclamación del comando `\index{}` porque aparecería en el título. En su lugar, podemos usar el comando `\IS`, que actúa como una separación para el índice pero que no aparece en el título.

```
1 \begin{theorem}[Teorema\IS de las
   gallinas cluecas]
2 Teorema.
3 \end{theorem}
```

Teorema III.2 (Teorema de las gallinas cluecas). *Teorema.*

Si, por lo que sea, queremos especificar nosotros el índice, podemos añadir un segundo argumento.

```
1 \begin{theorem}[Teorema de las gallinas
   cluecas][Mi entrada]
2 Teorema.
3 \end{theorem}
```

Teorema III.3 (Teorema de las gallinas cluecas). *Teorema.*

Hay definidos varios entornos similares a teoremas. Todos ellos se pueden titular poniendo entre corchetes el título.

```
1 \begin{lemma}[Lema cuarenta y dos]
2 \end{lemma}
3
4 \begin{corol}[Coro]
5 \end{corol}
6
7 \begin{prop}[De las desigualdades]
8 \end{prop}
9
10 \begin{axiom}
11 \end{axiom}
12
13 \begin{proof}
14 Probado queda.
15
16 Al final agrega un cuadradito de QED.
17 \end{proof}
18
19 \begin{op}{Nombre de operacin}
20 x = 3 + 1
21 \end{op}
```

Lema III.4 (Lema cuarenta y dos).

Corolario III.5 (Coro).

Proposición III.6 (De las desigualdades).

Axioma III.7.

Demostración. Probado queda.

Al final agrega un cuadradito de QED. □

NOMBRE DE OPERACION

$$x = 3 + 1$$

El entorno op incluye el modo matemático directamente, y el nombre de operación es obligatorio.

III.3.1.2. Definiciones y conceptos

Por otra parte tenemos el entorno defn para definiciones, que funciona exactamente igual que theorem:

```
1 \begin{defn}[Titulo de la definicin]([
   opcional) entrada para el ndice]
2 \end{defn}
```

Definición III.1 **Titulo de la definicin.**

III.3.1.3. Ejercicios

Hay dos entornos para incluir ejercicios o ejemplos en los archivos. El primero es example, que acepta un argumento opcional como nombre del ejemplo.

```
1 \begin{example}[Titulo]
2 Un ejemplo sobre cosas matematicas.
3
4 \[ 3 = 3 \]
5
6 Fin del ejemplo.
7 \end{example}
```

Ejemplo: Titulo *Un ejemplo sobre cosas matematicas.*

$$3 = 3$$

Fin del ejemplo.

El otro sería *problem*, que viene con bastantes cosillas. Acepta un argumento opcional, que es el número de ejercicio. Si no se pasa este argumento, la numeración es automática y se genera de la forma *Sección.Ejercicio*, de tal forma que se pueda referenciar luego con etiquetas. El comando `\solution` separa enunciado y solución, y los comandos `\ppart` y `\spart` sirven para separar los diferentes apartados (la numeración es automática). Además, hay una variante *problemS* que pone ejercicios sin la numeración por sección. Ejemplo:

```

1 \begin{problem}[3] \label{ej:Ejemplo}
2 Sea  $x$  una cosa, entonces calcula:
3
4 \ppart Calcula la divergencia de  $x$ .
5 \ppart Calcula 3.
6 \ppart Supongamos que  $x$  es otra cosa
   distinta. Entonces dime que te
   parece
7
8 \[ x^2 + 1 \]
9
10 \solution
11
12 \spart La divergencia es 0.
13
14 \spart
15
16 \[ 3 = \int_0^1 \delta(x^2) \]
17
18 \spart Muy bien.
19
20 \end{problem}
21
22 \begin{problemS}
23 Siguiendo con el ejercicio \ref{ej:
   Ejemplo}, dime mas cosas
24
25 \solution
26
27 \spart Mas cosas del \ref{ej:Ejemplo}.
28 \end{problemS}

```

Ejercicio 3.3: Sea x una cosa, entonces calcula:

- a) Calcula la divergencia de x .
- b) Calcula 3.
- c) Supongamos que x es otra cosa distinta. Entonces dime que te parece

$$x^2 + 1$$

APARTADO A)

La divergencia es 0.

APARTADO B)

$$3 = \int_0^1 \delta(x^2)$$

APARTADO C)

Muy bien.

Ejercicio 4: Siguiendo con el ejercicio ??, dime mas cosas

APARTADO A)

Mas cosas del ejercicio ??.

III.3.1.4. Imágenes

Hay dos comandos para poner fácilmente imágenes. El principal es `easyimgw`

```

1 \easyimgw{Patata.jpg}{Leyenda}{
   lblEtiqueta}{0.3}

```

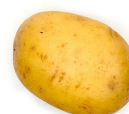


Figura III.1: Leyenda

El último argumento es la anchura de la imagen expresada como proporción de la

anchura del texto. 0.3 significa que ocupa un 30 % de la anchura del texto, por ejemplo.

También está el comando `\easyimg`, el uso es el mismo salvo que sólo necesita tres argumentos: la anchura se omite y se toma el valor por defecto del 80 % de anchura del texto.

III.3.1.5. Comandos adicionales

Dado que en cada asignatura hay varias cosas que se repiten, viene muy bien definir los comandos en *exmath.sty* para poder usarlos, y además así poder reutilizarlos en otras asignaturas.

Como son muchos, en esta documentación aparecen en el apéndice A, con ejemplos de uso y su salida. La tabla se genera de forma automática con el script *commandssampler.py*, en el directorio *Cosas Guays LaTeX*. La documentación de cómo funciona está en el propio paquete *exmath.sty*, aunque en algún momento debería de poner aquí cómo funciona.

III.3.2. *fancysprefs.sty*: Referencias mejoradas

fancysprefs está basado en *fancyref*, que ya hemos descrito en la sección II.2.5. Este paquete provee dos comandos adicionales, `\nref{nombreEtiqueta}` y `\nlref{nombreEtiqueta}`, además de traducir las cadenas de *fancyref* al castellano.

El comando `nref` hace algo parecido a *fref*, pero incluye también el nombre de lo que estemos referenciando. Así, sólo nos hace falta poner `\nref{sec:fancysprefs}` para que aparezca “*fancysprefs.sty*: Referencias mejoradas (III.3.2)”. Obviamente, el nombre cambiará automáticamente aquí si lo cambiamos en el entorno correspondiente. La variante `nlref` hace exactamente lo mismo pero con el título en minúsculas (muestra: *fancysprefs.sty*: Referencias mejoradas (III.3.2)), que viene bien cuando estamos referenciado definiciones y no queremos que la primera letra esté en mayúscula.

III.3.3. *MathUnicode.sty*: Letras griegas directamente desde el teclado

Escribir letras griegas o símbolos en L^AT_EX puede ser un poco rollo. Este paquete (incluido por defecto en la clase *apuntes* y *ejercicios*) permite hacerlo directamente desde el teclado. Así, en lugar de escribir `\alpha`, por ejemplo, simplemente pulsaríamos *Alt* + *A* para poner α en el texto y que L^AT_EX lo interprete sin fallos.

El paquete no soporta sólo letras griegas: también símbolos como $\forall, \exists, \partial, \in, \subset, \mathbb{R}, \mathbb{C}, \mathbb{N}, \geq, \notin$ y unos cuantos más. Para usarlo bien, probablemente tengas que cambiar la distribución de teclado de tu ordenador y así poder introducir estos caracteres. Para Mac, es bastante sencillo, para Linux hay que tocar alguna cosa más.

III.3.4. *tikztools.sty*: Herramientas y comandos extra para Tikz

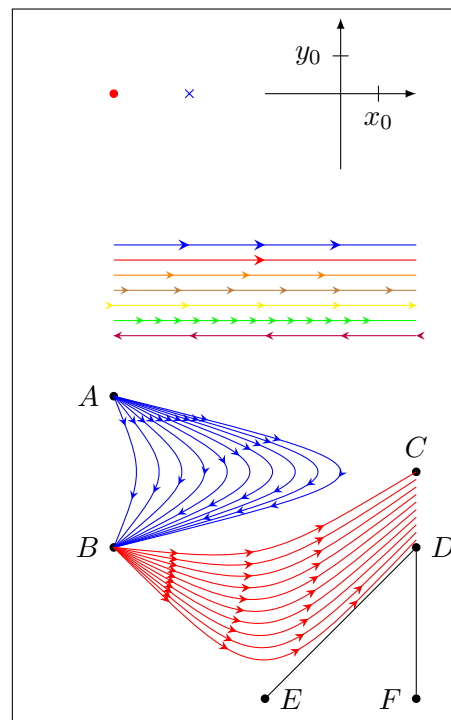
Para facilitar los dibujos de Tikz (ver sección II.2.7 para una introducción sobre qué es Tikz), el paquete *tikztools.sty* tiene varios comandos predefinidos. Incluye las librerías usuales, algunos estilos interesantes: *nodepoint* para poner puntos sencillos, *vnlin*, *hnl* para poner marcas verticales u horizontales, varias clases para marcar con flechas

líneas (*directed*, *dense directed*, *etc*, todas con un nombre bastante claro); un comando para dibujar familias de curvas (`\curvefam`). También está el comando `tikzangle` para marcar ángulos en LaTeX.

```

1 \begin{tikzpicture}
2 \node[nodepoint, red] at (0,0) {};
3 \node[cross, blue] at (1,0) {};
4
5 % vlnlin y hlnlin vienen bien para marcar puntos
   en ejes
6 \draw[->] (2,0) -- (4,0);
7 \draw[->] (3,-1) -- (3,1);
8
9 \node[vnlin, label=below:{$x_0$}] at (3.5, 0)
   {};
10 \node[hlnlin, label=left:{$y_0$}] at (3, 0.5) {};
11
12 % Los comandos "directed" nos permiten marcar
   flechas de flujo con distintas densidades
13 \draw[sparse directed, blue] (0, -2) -- (4, -2);
14 \draw[directed, red] (0, -2.2) -- (4, -2.2);
15 \draw[dense directed, orange] (0, -2.4) -- (4,
   -2.4);
16 \draw[plot directed, brown] (0, -2.6) -- (4,
   -2.6);
17 \draw[full directed, yellow] (0, -2.8) -- (4,
   -2.8);
18 \draw[really dense directed, green] (0, -3) --
   (4, -3);
19
20 % Todos los estilos tienen versin "invert"
21 \draw[invert full directed, purple] (0, -3.2) --
   (4, -3.2);
22
23 % Tambien hay familias de curvas (ver
   documentacin completa en tikztools.sty)
24 \node[nodepoint, label=left:{$A$}] (A) at (0,
   -4) {};
25 \node[nodepoint, label=left:{$B$}] (B) at (0,
   -6) {};
26 \node[nodepoint, label=above:{$C$}] (C) at (4,
   -5) {};
27 \node[nodepoint, label=right:{$D$}] (D) at (4,
   -6) {};
28 \node[nodepoint, label=right:{$E$}] (E) at (2,
   -8) {};
29 \node[nodepoint, label=left:{$F$}] (F) at (4,
   -8) {};
30
31 % Curvas de A a B que son "tiradas" por C.
32 \curvefam[blue, dense directed, thin]{(A)}{(B)}
   {(C)}
33
34 % Tambien se puede hacer que las lineas no
   lleguen todas al mismo punto
35 % Igualmente, se puede hacer con la salida con
   boffset/btarget
36 \curvefam[red, dense directed, thin, eoffset =
   0.1, etarget = C]{(B)}{(D)}{(E)}
37
38 % Se pueden marcar angulos
39 \draw (E) -- (D) -- (F);
40 \tikzangle[name={$\alpha$}, blue]{D}{E}{F}
41 \end{tikzpicture}

```



III.3.5. *fastbuild.sty*: Recortando los tiempos de compilación

Tikz es lento, y cuando hay un montón de dibujos recompilar un documento puede ser un infierno. Este paquete, *fastbuild.sty*, permite activar una caché de dibujos Tikz, de tal forma que los dibujos se generan en un PDF y después se incluyen, de tal forma que no hay que regenerarlos en cada compilación. Para activar esta caché, hay que incluir el comando `\precompileTikz` en el preámbulo del documento.

Internamente, lo que hace LaTeX cuando ahora se encuentre un dibujo Tikz, es abrir un nuevo proceso que compile el dibujo y después incluirlo en el documento. Por seguridad, la configuración por defecto de LaTeX impide la creación de nuevos procesos, por lo que hay que pasar como argumento `-shell-escape` al comando de compilación. Toda la caché se guarda en el directorio *tikzgen*, hay que procurar que esté creado porque si no LaTeX se quejará con un error bastante críptico.

III.3.6. *apuntes.cls*: La base para crear documentos de apuntes

También hay un archivo llamado *apuntes.cls*, que provee la clase *apuntes*. Básicamente, lo único que hace es cambiar la fuente, ajustar la geometría e incluir el paquete *exmath*. De esta forma, lo único que hay que hacer para usar todo el paquete y clase es cambiar la clase del documento. Es decir, que la primera línea sea

```
\documentclass{apuntes}
```

Además, la clase genera la portada y la cabecera, sólo hay que configurar la fecha, título y autor con los siguientes comandos, que deben ir después de la definición de `documentclass`.

```
\author{Autor}
\date{Fecha}
\title{Título del documento}
```

III.3.6.1. Cabecera de páginas

La cabecera de cada página tiene el siguiente formato: *Título - Fecha - Organización* en la parte izquierda y *Autor* en la parte derecha. La organización (por defecto, UAM) se puede cambiar escribiendo `\renewcommand{\organization}{lo que sea}` en el preámbulo. Si se deja vacío (`\renewcommand{\organization}{}{}`) el paquete lo gestiona bien y el formato pasa a ser *Título - Fecha*.

Si por lo que sea la cabecera es demasiado larga y se solapan las partes izquierda y derecha (por ejemplo, cuando hay muchos autores) se puede pasar la opción *shorthead* al `documentclass` (esto es, escribiendo `\documentclass[shorthead]{apuntes}`) para la parte izquierda de la cabecera sólo tenga el título del documento.

Además, las cabeceras se pueden eliminar completamente escribiendo `\pagestyle{empty}` en el documento (o, si sólo se quiere eliminar para una página, `\thispagestyle{empty}`).

III.3.6.2. Portada

Por defecto, la clase *apuntes* crea una portada algo más interesante que la que suelen traer las clases de L^AT_EX por defecto (ver la portada de este manual para un ejemplo). La

idea es una portada para los apuntes que creamos, así que incluye cosas como la fecha de compilación (así estamos seguros de cuándo se han actualizado los apuntes) o un enlace al repositorio de Github. Todo esto se puede personalizar:

- La fecha de compilación se puede eliminar del documento pasando la opción *nobuilddate* al comando *documentclass*.
- El color se puede cambiar redefiniendo el color *titlepagecolor* en el preámbulo. Algunos ejemplos de cómo hacerlo:

```
\documentclass{apuntes}

% ...
\colorlet{titlepagecolor}{green} % Podemos usar nombres
\colorlet{titlepagecolor}{red!80!black} % O hacer combinaciones de colores

\definecolor{titlepagecolor}{rgb}{200,200,100} % O usar directamente RGB
\definecolor{titlepagecolor}{HTML}{A0B0D0} % O también códigos HTML

\begin{document}
%...
```

- La firma de la esquina inferior derecha se puede eliminar pasando el argumento *signature = false* a *titlesetup*, es decir, escribiendo `\titlesetup{signature = false}`¹ antes del comando `\maketitle`.

III.3.7. *ejercicios.cls*: documentos simples y entrega de ejercicios


Por si hace falta hacer documentos que sean únicamente ejercicios (probablemente para entregarlos), la clase *ejercicios* sirve para eso. Reduce un poco los márgenes, pone una portada más discreta y pequeña, cambia a un tipo de letra algo más compacto y quita los marcos enormes a los problemas. El resto de comandos se puede seguir usando normalmente.

En este caso también puede venir útil usar el entorno `problemS` en lugar de `problem` para que los ejercicios no tengan la numeración estilo *sección.ejercicio* (ver la sección III.3.1.3 para una descripción más detallada).

¹La razón por la que hay que poner `titlesetup` y no argumentos opcionales a `maketitle` es que hay un problema (que no sé cuál es) que hace que no pueda redefinir `maketitle` de tal forma que reciba argumentos opcionales, así que he hecho esta pequeña chapuza para poder personalizar la portada.

Apéndice A

Comandos

<code>\triple</code>	
<code>\hint{Sugerencia}</code>	Sugerencia: Sugerencia
<code>\mop{operacion}</code>	operacion
<code>\imgref{referencia}</code>	(<i>figura??</i>)
Comandos de lógica	
<code>\dimplies</code>	\iff
<code>\nimplies</code>	\nRightarrow
<code>\Or</code>	\vee
<code>\y</code>	\wedge
<code>\tq</code>	$/$
<code>\tlq</code>	tal que
<code>\wtf</code>	(?)
<code>\MT</code>	<i>MquinadeTuring</i>
Calculo	
<code>\deriv{f}{arg1}</code>	$\frac{df}{darg1}$
<code>\dpa{arg0}{arg1}</code>	$\frac{\partial arg0}{\partial arg1}$
<code>\rot</code>	rot
<code>\dv</code>	div
<code>\grad</code>	∇
<code>\img</code>	img
<code>\epsilon</code>	ε
<code>\bigzero</code>	0
<code>\tlps</code>	$\tilde{\Psi}$
<code>\invers{arg0}</code>	$arg0^{-1}$
<code>\f</code>	f^{-1}
<code>\F</code>	F^{-1}
<code>\Rn xm</code>	$\mathbb{R}^M \times \mathbb{R}^N$
<code>\acum{arg0}</code>	$\#1 \subset (arg0)$
<code>\rnk</code>	\mathbb{R}^{N+K}
<code>\rk</code>	\mathbb{R}^{N+K}
<code>\ncl{f}</code>	$\ker f$

Análisis matemático	
<code>\normp{vec}{norma-n}</code>	$\ \vec{e}\ _{norma-n}$
<code>\liminf{variable}</code>	$\lim_{variable \rightarrow \infty}$
<code>\mylim{vec}{to}{var}</code>	$\lim_{\vec{e} \rightarrow \vec{t_0}} F(var)$
<code>\df{x,y,z}</code>	$dx \wedge dy \wedge dz$
<code>\dfl{comienzo}{fin}</code>	$dcomienzo \wedge \dots \wedge dfin$
<code>\id{x,y,z}</code>	$dx dy dz$
<code>\pb[T]</code>	T^*
Galois	
<code>\units{arg0}</code>	$(\mathbb{Z}/arg0\mathbb{Z})^*$
Investigación operativa:	
<code>\convx{arg0}</code>	$\text{conv}(arg0)$
<code>\afin{arg0}</code>	$\text{afin}(arg0)$
<code>\Hf{arg0}</code>	$\text{Hf}(arg0)$
<code>\epigraph{arg0}</code>	$\text{epi}(arg0)$
<code>\gvx</code>	\bar{x}
Notación de puntos de \mathbb{R}^N	
<code>\gx</code>	\bar{x}
<code>\gs</code>	\bar{s}
<code>\gy</code>	\bar{y}
<code>\gz</code>	\bar{z}
<code>\ga</code>	\bar{a}
<code>\gb</code>	\bar{b}
<code>\gv</code>	\bar{v}
<code>\gu</code>	\bar{u}
<code>\gw</code>	\bar{w}
Caracteres	
<code>\qeq</code>	$\stackrel{?}{=}$
<code>\r{A}</code>	
<code>\intr{A}</code>	$\overset{\circ}{A}$
Operadores	

<code>\comb{arg0}{arg1}</code>	$\binom{arg0}{arg1}$
<code>\abs{arg0}</code>	$ arg0 $
<code>\inv{arg0}</code>	$arg0^{-1}$
<code>\conj{arg0}</code>	$\overline{arg0}$
<code>\avg{arg0}</code>	$\overline{arg0}$
<code>\card{arg0}</code>	$ arg0 $
<code>\no{arg0}</code>	$\overline{arg0}$
<code>\gor{arg0}</code>	$\overline{arg0}$
<code>\floor{arg0}</code>	$\lfloor arg0 \rfloor$
<code>\argmin</code>	arg min
<code>\argmax</code>	arg max
<code>\sign</code>	sign
<code>\eq</code>	=
<code>\rango</code>	rango
<code>\rg</code>	Rg
<code>\tr</code>	traza
<code>\dist</code>	d
<code>\inter</code>	int
<code>\eqexpl[=]{arg1}</code>	$\underline{\underline{arg1}}$
<code>\eqreason[=]{arg1}</code>	$\begin{array}{c} = \\ \uparrow \\ arg1 \end{array}$
<code>\eqreasonup[=]{arg1}</code>	$\begin{array}{c} arg1 \\ \downarrow \\ = \end{array}$
Comandos para conjuntos y relaciones	
<code>\x</code>	\times
<code>\appl{f}{from}{to}</code>	$f : from \mapsto to$
<code>\uexists</code>	$\exists!$
<code>\sint</code>	\int_b^a
<code>\stdf</code>	$f : X \mapsto Y$
<code>\rel</code>	\mathcal{R}
<code>\parts{arg0}</code>	$\mathcal{P}(arg0)$

<code>\ind</code>	\mathbb{I}
Estadística	
<code>\esp[param]{arg1}</code>	$E_{param}(arg1)$
<code>\prob[param]{arg1}</code>	$P_{param}(arg1)$
<code>\var[param]{arg1}</code>	$V_{param}(arg1)$
<code>\fd</code>	\mathbb{F}
<code>\convstxt.arriba[txt. arriba]{var}</code>	$\xrightarrow[txt.arriba]{var \rightarrow converge}$
<code>\convdist[variable]</code>	$\xrightarrow[variable \rightarrow \infty]{d}$
<code>\convprob[variable]</code>	$\xrightarrow[variable \rightarrow \infty]{P}$
<code>\convcs[variable]</code>	$\xrightarrow[variable \rightarrow \infty]{c.s}$
<code>\sample[var][n]</code>	var_1, \dots, var_n
<code>\sesgo</code>	sesgo
<code>\ECM</code>	ECM
<code>\emv</code>	emv
<code>\cov{arg0}</code>	$cov(arg0)$
<code>\corr{arg0}</code>	$corr(arg0)$
Abreviaciones para conjuntos usuales	
<code>\nat</code>	\mathbb{N}
<code>\ent</code>	\mathbb{Z}
<code>\rac</code>	\mathbb{Q}
<code>\real</code>	\mathbb{R}
<code>\cplex</code>	\mathbb{C}
<code>\vec{arg0}</code>	$\mathbf{arg0}$
Vectores	
<code>\pesc{arg0}</code>	$\langle arg0 \rangle$
<code>\md{arg0}</code>	$\ arg0\ $
Vectores usuales, para no tener que andar escribiendo	
<code>\vx</code>	\mathbf{x}
<code>\vy</code>	\mathbf{y}
<code>\vf</code>	\mathbf{F}
<code>\vg</code>	\mathbf{G}

<code>\va</code>	a
<code>\vb</code>	b
<code>\vc</code>	c
<code>\vu</code>	u
<code>\vn</code>	n
<code>\vv</code>	v
<code>\ve</code>	e
<code>\vr</code>	r
<code>\vz</code>	z
Complejos	
<code>\ctrig{angulo}</code>	$\cos(angulo) + i \sin(angulo)$
<code>\ceul{mod}{angulo}</code>	$mode^{iangulo}$
<code>\i</code>	β
Matrices	
<code>\trans{arg0}</code>	$arg0^T$
Probabilidad	
<code>\bin</code>	Bin
<code>\geom</code>	Geom
Estructuras algebraicas	
<code>\gen{genlist}</code>	$\langle genlist \rangle$
<code>\ord</code>	ord
<code>\gr</code>	gr
<code>\kbb</code>	\mathbb{K}
Geometría de curvas y superficies	
<code>\mv{vec}</code>	vec
<code>\cv[\alpha]</code>	k_α
<code>\cvv[\alpha]</code>	\mathbf{k}_α
<code>\cn[\alpha]</code>	\mathbf{n}_α
<code>\ct[\alpha]</code>	\mathbf{t}_α
<code>\cb[\alpha]</code>	\mathbf{b}_α
<code>\ctr[\alpha]</code>	τ_α
<code>\wein</code>	\mathcal{W}

<code>\lfi</code>	\mathcal{L}
<code>\vV</code>	V
Topología	
<code>\dst</code>	d
<code>\bola</code>	\mathbb{B}
<code>\sdst</code>	(X, d)
<code>\topl</code>	\mathcal{T}
<code>\stopl</code>	(X, \mathcal{T})
<code>\base</code>	\mathcal{B}
<code>\toplb</code>	$\mathcal{T}_\mathcal{B}$
<code>\adh{arg0}</code>	$\overline{arg0}$
<code>\iffdef</code>	$\stackrel{\text{def}}{\iff}$
<code>\bolac</code>	$\mathring{\mathbb{B}}$
<code>\tops</code>	(X, \mathcal{T})
<code>\topu</code>	\mathcal{T}_{usual}
<code>\quot{arg0}{arg1}</code>	$arg0 / arg1$
<code>\dfm</code>	\simeq_p
<code>\restr{arg0}{arg1}</code>	$arg0 _{arg1}$
<code>\bbs</code>	\mathbb{S}
<code>\crc[1]</code>	\mathbb{S}^1
<code>\torus[2]</code>	\mathbb{T}^2
<code>\disc</code>	\mathbb{D}
TIM	
<code>\algb{C}</code>	\mathcal{C}
<code>\algbA</code>	\mathcal{A}
<code>\algbM</code>	\mathcal{M}
<code>\algbC</code>	\mathcal{C}
<code>\algbB</code>	\mathcal{B}
<code>\algbT</code>	\mathcal{T}
<code>\algbP</code>	\mathcal{P}
<code>\salgb</code>	σ -álgebra
<code>\sfin</code>	σ -finita

Galois	
<code>\gal</code>	Gal
Geometría	
<code>\tens</code>	\mathcal{T}
<code>\dif</code>	d
<code>\Dif</code>	D
<code>\projp</code>	\mathbb{RP}
<code>\projcp</code>	\mathbb{CP}
<code>\lie{arg0}</code>	$[arg0]$
<code>\tgs</code>	T
<code>\tgds</code>	T^*
<code>\inmr{arg0}{arg1}{arg2}</code>	$arg0 : arg1 \hookrightarrow arg2$
<code>\halfp</code>	\mathbb{H}
<code>\hash</code>	$\#$
<code>\difp[F][p]</code>	$DF _p$
Variable Compleja	
<code>\cind</code>	$\mathbb{I}nd$
<code>\Re</code>	Re
<code>\Im</code>	Im
<code>\Res</code>	Res
Modelización	
<code>\fe{arg0}</code>	$e(arg0)$
Variable real	
<code>\lebg</code>	\mathcal{L}
<code>\borel</code>	\mathcal{B}
<code>\meds[X]</code>	(X, \mathcal{X})
<code>\meas[X][\mu]</code>	(X, \mathcal{X}, μ)
<code>\minuszero</code>	$\setminus \{0\}$
<code>\essup</code>	essup
<code>\esinf</code>	esinf
<code>\convsmcd</code>	\xrightarrow{m}
<code>\mpos</code>	\mathcal{M}_+
<code>\msgn</code>	\mathcal{M}

<code>\mfin</code>	\mathcal{M}_F
<code>\sop</code>	sop
<code>\essop</code>	essop
<code>\esp1loc[1][\real^N]</code>	$\mathcal{L}_{\text{loc}}^1(\mathbb{R}^N)$
<code>\esp1[1][\real^N]</code>	$\mathcal{L}^1(\mathbb{R}^N)$
<code>\espLloc[1][\real^N]</code>	$L_{\text{loc}}^1(\mathbb{R}^N)$
<code>\espL[1][\real^N]</code>	$L^1(\mathbb{R}^N)$
<code>\espLp</code>	L^p
<code>\esplp</code>	\mathcal{L}^p
<code>\pnorm{arg0}</code>	$\ arg0\ _p$
<code>\espell[2]</code>	$\ell^2(\mathbb{N})$
<code>\spn</code>	span
<code>\densein</code>	$\xhookrightarrow{\text{denso}}$
<code>\fourier</code>	\mathfrak{F}
Análisis funcional	
<code>\linapp[X][Y]</code>	$\mathcal{L}(X, Y)$
<code>\linend[H]</code>	$\mathcal{L}(H)$
<code>\hconvs[txt. arriba]{arg1}{arg2}</code>	$\xrightarrow[n \rightarrow \infty]{\text{txt. arriba}} arg1 arg2$
<code>\wconvs[][\infty]</code>	\rightharpoonup
<code>\wsconvs[][\infty]</code>	$\overset{*}{\rightharpoonup}$
<code>\dual[X]</code>	X^*
<code>\ddual[X]</code>	X^{**}

Índice alfabético

-shell-escape, 15, 39
 \alph, Alph, 20
 \arabic, 20
 \begin{figure}, 8
 \begin{tabular}, 9
 \begin{tikzpicture}, 22
 \begin{verbatim}, 7
 \caption, 9
 \chapter, 7
 \curvefrom, 36
 \def, 29
 \displaystyle, 16
 \documentclass, 5
 \footnote, 10
 \footnotemark, 10
 \footnotetext, 10
 \fref, 20
 \hline, 9
 \includegraphics, 8
 \input, 19
 \label, 9, 10
 \left, 12
 \leftmark, 21
 \let, 29
 \newcommand, 27
 \newcommandx, 28
 \newenvironment, 27
 \newenvironmentx, 28
 \newif, 30
 \nref, 36
 \nrefl, 36
 \ref, 10
 \renewcommand, 28
 \right, 12
 \rightmark, 21
 \roman, Roman, 20
 \tikzangle, 36
 \titlesetup, 40
 \usepackage, 5
 \verb, 7
 3D, 25

babel, 22
 Beamer, 26
 Cabeceras, 21
 commandsampler.py, 36
 Cosas guays LaTeX, 32
 Cursiva, 7
 cwlcreator.py, 33
 fancyhdr, 21
 fancyref, 20, 36
 fancyspreads, 36
 fastbuild.sty, 39
 Fracción, 12
 geometry, 19
 Imagen, 8
 install, 32
 latexmk, 25
 Lista, 8
 Margen, 19
 MathUnicode.sty, 36
 Negrita, 7
 nobuilddate, 40
 Párrafo, 7
 pgfkeys, 28
 pgfplots, 24
 Pie de página, 21
 problem, 35
 problemS, 35
 Raíz, 12
 Sección, 6
 shorthead, 39
 Tabla, 9
 de contenidos, 6
 texdoc, 18
 TeXStudio, 4

Tikz, [22](#)
tikz-3dplot, [25](#)
tikztools.sty, [36](#)
tocdepth, [20](#)

xargs, [28](#)