

---

# Control Remoto de Videojuegos con Smartphones

---



## MEMORIA DE TRABAJO DE FIN DE GRADO

Pablo Gómez Calvo  
Sergio J. Higuera Velasco

Grado de Desarrollo de Videojuegos  
Facultad de Informática  
Universidad Complutense de Madrid

Mayo 2019

Documento maquetado con TEXIS v.1.0+.

Este documento está preparado para ser imprimido a doble cara.

# Control Remoto de Videojuegos con Smartphones

*Memoria de Trabajo de Fin de Grado*  
**Grado de Desarrollo de Videojuegos**  
**Mayo 2019**  
**Director: Carlos León Aznar**  
**Co-Director: Pedro Pablo Gómez Martín**

*Versión 1.0+*

**Grado de Desarrollo de Videojuegos**  
**Facultad de Informática**  
**Universidad Complutense de Madrid**

**Mayo 2019**

Copyright © Pablo Gómez Calvo y Sergio J. Higuera Velasco

*Al duque de Béjar  
y  
a tí, lector carísimo*



*I can't go to a restaurant and  
order food because I keep looking  
at the fonts on the menu.*

*Donald Knuth*



# Agradecimientos

*A todos los que la presente vieran y entendieren.*

Inicio de las Leyes Orgánicas. Juan Carlos I

Groucho Marx decía que encontraba a la televisión muy educativa porque cada vez que alguien la encendía, él se iba a otra habitación a leer un libro. Utilizando un esquema similar, nosotros queremos agradecer al Word de Microsoft el habernos forzado a utilizar L<sup>A</sup>T<sub>E</sub>X. Cualquiera que haya intentado escribir un documento de más de 150 páginas con esta aplicación entenderá a qué nos referimos. Y lo decimos porque nuestra andadura con L<sup>A</sup>T<sub>E</sub>X comenzó, precisamente, después de escribir un documento de algo más de 200 páginas. Una vez terminado decidimos que nunca más pasariamos por ahí. Y entonces caímos en L<sup>A</sup>T<sub>E</sub>X.

Es muy posible que hubiéramos llegado al mismo sitio de todas formas, ya que en el mundo académico a la hora de escribir artículos y contribuciones a congresos lo más extendido es L<sup>A</sup>T<sub>E</sub>X. Sin embargo, también es cierto que cuando intentas escribir un documento grande en L<sup>A</sup>T<sub>E</sub>X por tu cuenta y riesgo sin un enlace del tipo “*Author instructions*”, se hace cuesta arriba, pues uno no sabe por donde empezar.

Y ahí es donde debemos agradecer tanto a Pablo Gervás como a Miguel Palomino su ayuda. El primero nos ofreció el código fuente de una programación docente que había hecho unos años atrás y que nos sirvió de inspiración (por ejemplo, el fichero `guionado.tex` de T<sub>E</sub>X<sup>I</sup>S tiene una estructura casi exacta a la suya e incluso puede que el nombre sea el mismo). El segundo nos dejó husmear en el código fuente de su propia tesis donde, además de otras cosas más interesantes pero menos curiosas, descubrimos que aún hay gente que escribe los acentos españoles con el \’{\i}.

No podemos tampoco olvidar a los numerosos autores de los libros y tutoriales de L<sup>A</sup>T<sub>E</sub>X que no sólo permiten descargar esos manuales sin coste adicional, sino que también dejan disponible el código fuente. Estamos pensando en Tobias Oetiker, Hubert Partl, Irene Hyna y Elisabeth Schlegl, autores del famoso “The Not So Short Introduction to L<sup>A</sup>T<sub>E</sub>X<sup>2</sup><sub>\varepsilon</sub>” y en Tomás

Bautista, autor de la traducción al español. De ellos es, entre otras muchas cosas, el entorno `example` utilizado en algunos momentos en este manual.

También estamos en deuda con Joaquín Ataz López, autor del libro “Creación de ficheros L<sup>A</sup>T<sub>E</sub>X con GNU Emacs”. Gracias a él dejamos de lado a WinEdt y a Kile, los editores que por entonces utilizábamos en entornos Windows y Linux respectivamente, y nos pasamos a emacs. El tiempo de escritura que nos ahorramos por no mover las manos del teclado para desplazar el cursor o por no tener que escribir `\emph` una y otra vez se lo debemos a él; nuestro ocio y vida social se lo agradecen.

Por último, gracias a toda esa gente creadora de manuales, tutoriales, documentación de paquetes o respuestas en foros que hemos utilizado y seguiremos utilizando en nuestro quehacer como usuarios de L<sup>A</sup>T<sub>E</sub>X. Sabéis un montón.

Y para terminar, a Donald Knuth, Leslie Lamport y todos los que hacen y han hecho posible que hoy puedas estar leyendo estas líneas.

# Resumen

*Desocupado lector, sin juramento me  
podrás creer que quisiera que este libro  
[...] fuera el más hermoso, el más  
gallardo y más discreto que pudiera  
imaginarse.*

Miguel de Cervantes, Don Quijote de la  
Mancha

TEX!S es un conjunto de ficheros L<sup>A</sup>T<sub>E</sub>X que pueden servir para escribir tesis doctorales, trabajos de fin de master, de fin de carrera y otros documentos del mismo estilo. El documento que tienes en tus manos es un manual que explica las distintas características de la plantilla. En los distintos capítulos iremos explicando los ficheros existentes en TEX!S así como su función. También se explican algunas de las características, como por ejemplo ciertos comandos que facilitan la escritura de los documentos.

Aunque el código L<sup>A</sup>T<sub>E</sub>X utilizado en TEX!S está muy comentado para su uso fácil, creemos que las explicaciones que aquí se proporcionan pueden ser útiles.

Hay dos distribuciones distintas de TEX!S: el código fuente completo de este manual (de forma que TEX!S es “*su propio manual*”<sup>1</sup>), o una distribución casi “vacía de contenido”, que tiene un único capítulo y apéndice vacío, pero mantiene la portada, dedicatoria, agradecimientos y bibliografía del manual.

Dependiendo, pues, de qué distribución escojas, partirás directamente de los ficheros .tex de este manual y eliminarás su texto para añadir el tuyo, o de un conjunto de ficheros sin apenas contenido que rellenarás. Aconsejamos esta última aproximación por ser más cómoda. Sin embargo, hacemos disponible los ficheros .tex del manual como referencia.

Para facilitar las cosas, hemos intentado que su estructura sea parecida a la de una posible tesis. De esta forma el código fuente del propio manual puede servir como punto de partida para la escritura de este tipo de documentos. Como podrás comprobar, en algún momento nos ha sido difícil justificar la existencia de ciertos elementos pues no eran realmente relevantes para el ma-

---

<sup>1</sup>Los expertos en lógica seguro que tendrían algo que decir al respecto...

nual. En esos casos, piensa que están ahí no porque sean importantes desde el punto de vista de *este* documento, sino porque muy posiblemente estarían en el tipo de textos para los que TeXiS es útil.

Al estar compuesto por varios tipos de ficheros, TeXiS se rige por varias licencias:

La plantilla (ficheros en el directorio TeXiS) se distribuye bajo la *LATeX Project Public License* (Licencia Pública del Proyecto LATeX).



Los ficheros `Makefile` y scripts de apoyo a la generación del documento, se distribuyen bajo licencia GPLv3.



El *manual* de TeXiS se distribuye con una licencia Creative Commons (CC-BY-SA).

# Índice

<b>Agradecimientos</b>	<b>IX</b>
<b>Resumen</b>	<b>XI</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Introducción . . . . .	1
1.2. Proposición de la idea . . . . .	2
1.3. Estructura de capítulos . . . . .	2
<b>2. Estado del arte</b>	<b>3</b>
2.1. Herramientas para el desarrollo . . . . .	3
2.1.1. Unity . . . . .	3
2.1.2. Android Studio . . . . .	4
2.1.3. ZXing . . . . .	5
2.2. Proyectos Similares . . . . .	6
2.2.1. WiiU . . . . .	6
2.2.2. Joystick . . . . .	6
2.2.3. PlayLink . . . . .	6
<b>3. Objetivos y especificación</b>	<b>9</b>
3.1. Objetivos . . . . .	9
3.2. Plan de trabajo . . . . .	10
3.3. Metodologías . . . . .	11
3.4. Herramientas utilizadas . . . . .	11
<b>4. Gestión de las imágenes</b>	<b>13</b>
4.1. Introducción . . . . .	13
4.2. Gestión de imágenes . . . . .	14
4.3. Formato de las imágenes . . . . .	16
4.4. Imágenes independientes del programa generador . . . . .	17
4.5. Gestión de imágenes y control de versiones . . . . .	17
4.6. Imágenes divididas . . . . .	18

Notas bibliográficas . . . . .	22
En el próximo capítulo . . . . .	22
<b>5. Bibliografía y acrónimos</b>	<b>23</b>
5.1. Bibliografía . . . . .	23
5.1.1. Ficheros involucrados . . . . .	24
5.1.2. Referencias con <code>natbib</code> . . . . .	24
5.1.3. Modificaciones en los <code>@bibitem</code> . . . . .	25
5.1.4. Cambio del estilo de la bibliografía . . . . .	27
5.2. Acrónimos . . . . .	28
5.2.1. Acrónimos con <code>glosstex</code> . . . . .	28
5.2.2. Acrónimos en <code>TEXIS</code> . . . . .	32
5.2.3. Más allá de <code>TEXIS</code> . . . . .	33
Notas bibliográficas . . . . .	34
En el próximo capítulo . . . . .	34
<b>6. Makefile</b>	<b>35</b>
6.1. Introducción . . . . .	35
6.2. Objetivos del <code>Makefile</code> . . . . .	36
6.3. Funcionamiento interno . . . . .	38
6.3.1. La compilación de las imágenes . . . . .	38
6.3.2. Makefile, <code>GlossTeX</code> , y cambio de modo de generación .	40
Notas bibliográficas . . . . .	41
<b>I Apéndices</b>	<b>43</b>
<b>A. Así se hizo...</b>	<b>45</b>
A.1. Edición . . . . .	45
A.2. Encuadernación . . . . .	46
A.3. En el día a día . . . . .	46

# Índice de figuras

4.1.	Figura utilizada para marcar una imagen por hacer. . . . .	15
4.2.	Ejemplo de uso de <code>subfloat</code> . . . . .	21
5.1.	Resultado de la lista de acrónimos . . . . .	28
A.1.	Encuadernación y márgenes guillotinados . . . . .	47
A.2.	Servidor de integración continua . . . . .	48



# Índice de Tablas

4.1. Formatos de imágenes para <code>latex</code> y <code>pdflatex</code> . . . . .	16
5.1. Distintas opciones de referencias con <code>natbib</code> . . . . .	25



# Capítulo 1

## Introducción

**RESUMEN:** Este capítulo presenta una breve introducción al proyecto. El lector podrá hacerse una idea de qué es y para qué sirve. También se encuentra aquí una descripción del resto de capítulos del manual.

### 1.1. Introducción

La industria del videojuego en esta última década nos ha enseñado que la innovación a la hora de crear experiencias nuevas para los usuarios es algo que enriquece a muchos jugadores, estamos ayudando a que nuestro jugador se meta más en el papel que tiene que desempeñar dentro del videojuego que esté jugando.

Es por esta razón que muchas empresas, cada vez más, dedican gran cantidad de sus recursos a hacer realidad muchas experiencias que los usuarios quieren tener como, por ejemplo, jugar a juegos deportivos mientras se hace deporte.

Pero para invertir en crear este nuevo tipo de estilos de juego se necesita una gran financiación, cosa que estudios pequeños no tienen. La mayor fuente de ideas para estos proyectos de creación de nuevas experiencias para los usuarios suelen ser los estudios independientes ya que se suele trabajar en un ámbito donde las ideas son más dadas a aflorar. Estos estudios independientes usan motores como Unity3D o Unreal Engine 4. Estos motores son gratuitos y ofrecen una gran cantidad de herramientas a disposición de sus usuarios para que los desarrolladores puedan ahorrar tiempo de implementación de una nueva feature y lo utilicén para que su juego siga creciendo. El desarrollo

para dispositivos móviles en estudios independientes ha crecido de manera exponencial gracias a que motores como Unity lo hacen bastante accesible.

Con todos estos ingredientes nos hicimos la siguiente pregunta, ¿Y si un juego se pudiese jugar en el teléfono móvil pero en verdad el juego se estuviese ejecutando en el ordenador?

## 1.2. Proposición de la idea

Este trabajo pretende contribuir al crecimiento de las experiencias que se le puede ofrecer a un usuario desde un juego de una empresa independiente que utilice Unity como su motor de desarrollo. Se busca crear un híbrido entre un juego convencional de ordenador junto con una tecnología, la cual la mayor parte de los usuarios tiene, como son los teléfonos móviles.

La herramienta propuesta se apoyará sobre la herramienta líder para desarrollar juegos de manera independiente, Unity, sirviendo como una prueba de concepto para integrar el uso de un dispositivo móvil a un juego convencional de ordenador.

Una vez terminada, la herramienta podrá ser usada por cualquier desarrollador que lo desee. La herramienta será de uso libre para su posible ampliación.

## 1.3. Estructura de capítulos

Este documento está estructurado en los siguientes capítulos:

- El capítulo 2 expone una revisión del estado del arte de las tecnologías involucradas en el desarrollo de este trabajo.
- El capítulo 3 se centra en los objetivos a cumplir en este proyecto junto con la especificación y las metodologías que se van a seguir para la completa realización de este trabajo.
- El capítulo 4 explica todo lo relacionado con el diseño de la herramienta y la posterior implementación de la misma. También incluye la explicación de los prototipos llevados a cabo durante el desarrollo del proyecto.
- El capítulo 5 aborda los resultados obtenidos y la demostración de lo conseguido.
- El capítulo 6 expone la discusión y las conclusiones.

# Capítulo 2

## Estado del arte

**RESUMEN:** Este capítulo explica las herramientas que han sido utilizadas en el desarrollo de este trabajo. También se odrecen unos cuantos ejemplos en los cuales nos hemos basado y nos han llevado a pensar que la idea de este proyecto es algo útil.

### 2.1. Herramientas para el desarrollo

En este proyecto se ha unificado el uso de varias tecnologías, en principio independientes, para formar una herramienta que une el desarrollo de videojuegos para ordenador y el uso de una aplicación de Android que nos facilite la conexión Pc <->Android. Estas tecnologías/herramientas han sido:

#### 2.1.1. Unity

Unity es un motor de videojuegos multiplataforma creado por Unity Technologies. Se encuentra disponible para sistemas Windows, Mac Os X y Linux. Es una de las herramientas de desarrollo de videojuegos más populares actualmente en el mundo de los desarrolladores independientes. También cuenta con una gran cantidad de documentación generada tanto por los diversos usuarios como por sus creadores.

Las principales características de Unity son:

- **Multiplataforma.** El hecho de que Unity sea un sistema multiplataforma, te garantiza poder hacer juegos/aplicaciones que puedan ejecutarse en cualquier dispositivo a un coste bastante bajo en cuanto a esfuerzo.

- **Extensibilidad.** Este es el aspecto en el cual nosotros más nos hemos fijado, ya que Unity permite la integración de Plug-ins de una manera bastante cómoda y ágil.
- **Potencia.** Unity cuenta con un gran competidor en este aspecto, Unreal Engine 4, pero en las últimas versiones se está intentando que esa gran diferencia que antes había entre Unity y Unreal sea cada vez más pequeña.
- **Documentación.** El caso de Unity en cuanto a documentación es algo que los desarrolladores de Unity cuidan bastante. Puedes acceder a la documentación de una manera muy intuitiva desde la misma página web de Unity y en los últimos meses están incluyendo proyectos de juegos completos de principio a fin.
- **Comunidad.** Debido a que Unity es un motor de videojuegos potente y gratuito, cuenta con un gran número de gente que sabe utilizarlo y crea contenido didáctico para que su aprendizaje sea más ameno.



### 2.1.2. Android Studio

Android Studio es el entorno de desarrollo oficial de la plataforma Android. Fue desarrollada por Google y sustituyó a Eclipse en el desarrollo de aplicaciones para dispositivos Android. Este IDE puede utilizarse tanto en Windows, Mac OS X y Linux pero únicamente puede usarse para el desarrollo de Android.

Las principales características de Android Studio son:

- **Específico.** Si el producto que quieras desarrollar va a ser exclusivo de un sistema Android, con Android Studio te vas a centrar en explotar las funcionalidades de ese sistema al máximo sin tener que lidiar con diferentes sistemas que no te interesan.
- **Prueba virtual.** Android Studio cuenta con una serie de Emuladores virtuales de dispositivos móviles para probar tus aplicaciones sin necesidad de tener el teléfono físico.

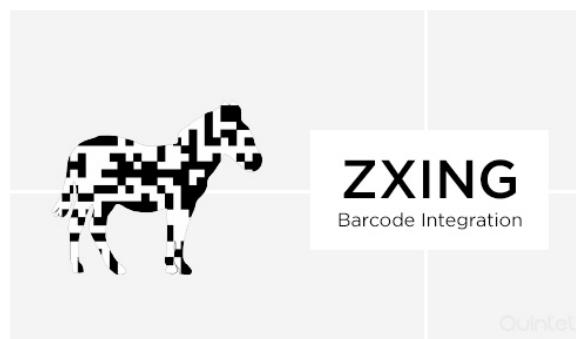
- **Editor de diseño.** Android Studio cuenta con un editor visual para poder acomodar el layout de tu aplicación Android de una manera mucho más sencilla.



### 2.1.3. ZXing

Este es un proyecto del tipo Open-Source. Esta librería de procesamiento de imágenes de códigos de barras y QR's está implementada en Java y tiene diferentes versiones en los distintos lenguajes. En nuestro caso hemos usando la versión de .NET para usarla desde Unity. Las principales características de ZXing son:

- **Versatilidad.** Al ser una librería que tiene soporte en muchos otros lenguajes que no son Java, puedes utilizarla en tus proyectos multiplataforma si estas plataformas usan diferentes lenguajes como en nuestro caso (Android Studio y Unity).
- **Facilidad.** Ponerse a trabajar con ello fue algo sencillo, también en los diferentes repositorios de GitHub hay demos de cómo usar la librería.



## 2.2. Proyectos Similares

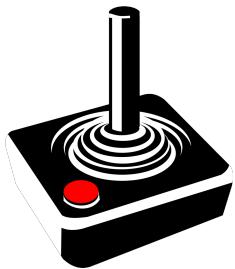
Hay algunas empresas que han invertido mucho en innovar y crear nuevas formas para que los usuarios disfruten de los diferentes juegos. Algunos de los ejemplos que ya existen en el mercado y han servido de inspiración para la realización de este proyecto son:

### 2.2.1. WiiU



La Wii U es una consola de Nintendo de la octava generación. Esta consola de sobremesa incluía una pantalla portátil que servía a la vez de pantalla secundaria y mando. Esta pantalla portátil es táctil y recibe una señal de 480p. A este nuevo mando se le denomina Wii U GamePad. Este nuevo mando permitía a los desarrolladores tener un HUD mucho más limpio dentro del juego ya que, en muchos casos, esta pantalla era utilizada para poner elementos como el minimapa.

### 2.2.2. Joystick



El joystick es un periférico de entrada que consiste en una palanca que tiene libre movimiento sobre una base. Este informa sobre su ángulo y dirección al dispositivo al que esté conectado. Los joysticks se han utilizado siempre en el mundo de los videojuegos, empezando por la Atari hasta llegar al DualShock de PlayStation que hay actualmente.

### 2.2.3. PlayLink

PlayStation es una de las compañías que más jugadores mueve, sus consolas son de las más vendidas en todo el mundo y para que esto sea posible siempre tienen que intentar estar a la cabeza de nuevos periféricos, nuevos juegos y, por supuesto, darles a sus usuarios las mejores experiencias posibles. En 2017, Sony PlayStation sacó al mercado una nueva serie de juegos llamados PlayLink. Estos juegos tienen una particularidad con respecto a un juego convencional de consola, estos juegos están hechos para jugarlos con gente y usando un dispositivo móvil como mando/controlador. Conectando la consola y el móvil a la misma red WIFI y conectándolos a través de una

aplicación, se pueden conectar de 2 a 8 jugadores, depende de los que admite cada juego, para poder jugar en familia o con amigos.





## Capítulo 3

# Objetivos y especificación

**RESUMEN:** Este capítulo se centra en la organización que se ha seguido para la realización del proyecto. También se explican los objetivos de este TFG y las metodologías utilizadas.

### 3.1. Objetivos

Con este proyecto se han querido tener los siguientes objetivos generales:

- Crear una librería que permitiese la utilización de un dispositivo móvil como gamepad y pantalla secundaria.
- Aprender todo lo posible sobre la implementación de una librería que posteriormente puede ser usada por cualquier desarrollador.
- Profundizar en el uso de la red y en la realización de un protocolo de conexión entre móvil y ordenador.
- Crear un proyecto a modo de demostración de lo que hemos sido capaces de desarrollar y demuestre las capacidades de la herramienta desarrollada.

En las próximas subsecciones plantearemos la metodología y el plan de trabajo seguido.

### 3.2. Plan de trabajo

Para la realización de este proyecto se han cogido y unificado muchos aspectos que se han planteado de manera separada durante todo el grado. Unity es algo que hemos utilizado bastante pero sin duda la parte en la que hemos tenido que dedicar más tiempo de investigación ha sido a cómo plantear un protocolo de comunicación y que este se adapte bien a la latencia de una red doméstica.

En los primeros meses del proyecto se asentaron las herramientas de gestión que iban a utilizarse. Se creó una organización en GitHub donde se crearon 2 repositorios, uno para las demos y el futuro proyecto final/demostación técnica. En otro repositorio se fueron subiendo los cambios realizados a la memoria del proyecto. Las herramientas fueron fijadas desde la primera reunión en Octubre de 2018.

1. Se decidió utilizar MiKTeX como editor de textos LaTeX.
2. Como sistema de control de versiones se utilizaría GitHub con los dos repositorios mencionados anteriormente.
3. Como herramienta de comunicación con los directores del proyecto se decidió utilizar correos electrónicos.
4. De manera interna para los miembros del proyecto, se utilizará un sistema de seguimiento de tareas para saber los objetivos a cumplir y el estado de cada tarea en curso.

Los recursos físicos para este proyecto serán 2 ordenadores con un sistema Windows instalado para ejecutar Unity y Android Studio correctamente. También se precisarán de al menos 1 dispositivo móvil para comprobar el funcionamiento correcto del proyecto.

En cuanto a reuniones, se acuerda una reunión cada 2 semanas tanto con los miembros del proyecto como con los directores. El objetivo de estas reuniones es comprobar lo que se ha avanzado en las 2 semanas de desarrollo, exponer ideas que ayuden a mejorar el proyecto y planificar las nuevas tareas que se presentarán tras las siguientes 2 semanas.

Se marcaron diferentes hitos a lo largo del proyecto:

1. **Reunión Pre-Navidad (19/12/2018).** El objetivo para esta reunión es tener una conexión establecida entre un dispositivo Android con un proyecto de Unity. Junto con esa conexión se pretende tener un personaje en movimiento con el Input que reciba del dispositivo móvil.
2. **Reunión Pre-Semana Santa (10/04/2019).** El objetivo para esta reunión es tener una cámara de Unity enviando imagen vía streaming

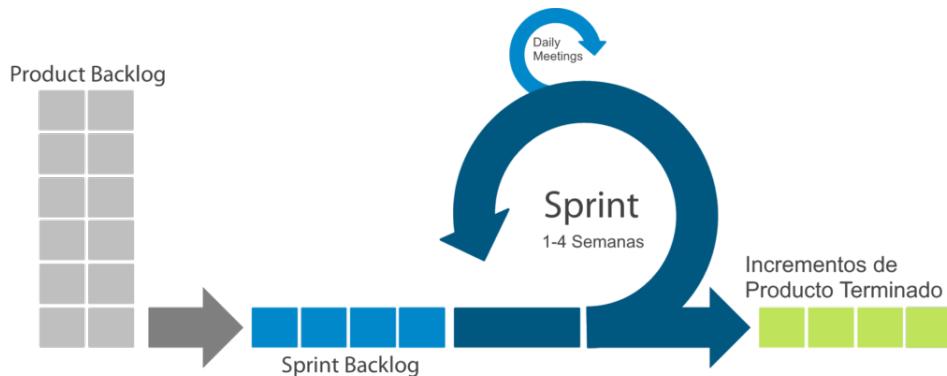
al dispositivo móvil mientras que se utiliza este mismo como mando/-controlador.

3. **Reunión Pre-Exámenes (22/05/2019).** En esta reunión se debe tener una demostración de las capacidades de la herramienta. Esta demostración será la que se presente ante el tribunal de calificación de TFG's.

### 3.3. Metodologías

La metodología utilizada para la realización de este proyecto está basada en el desarrollo ágil. Se han marcado pequeños objetivos de 2 semanas de duración llamados sprints hasta llegar a las metas grandes o hitos mencionados en el apartado anterior.

La metodología que se iba a usar estaba definida desde el primer día ya que es la que los miembros del proyecto siguen en todos los trabajos que realizan. Se decidió usar **Scrum**. Scrum es una metodología ágil que adopta una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del proyecto en cuestión. En las reuniones se veía hasta donde se había llegado y en base a eso se planificaba el trabajo para la siguiente reunión. Con constantes revisiones, se pueden ir añadiendo nuevas features al proyecto que en primera instancia no se contemplaron.



### 3.4. Herramientas utilizadas

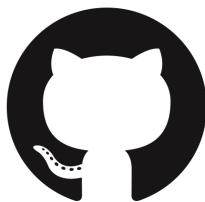
En la sección 3.1 se ha hablado de las herramientas que se acordaron utilizar para la realización de este proyecto:

1. **MiKTeX y LaTeX**



Las características más apreciables de MiKTeX son su habilidad de actualizarse por sí mismo descargando nuevas versiones de componentes y paquetes instalados previamente, y su fácil proceso de instalación. Posee compiladores TeX y LaTeX para generar archivos .pdf y herramientas para generar bibliografías e índices de una manera sencilla.

## 2. GitHub



GitHub es una plataforma de desarrollo cooperativo en la que se pueden alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza para la creación y almacenamiento de código fuente de manera pública. GitHub ofrece varias características entre las que destacan:

- Wiki para cada proyecto.
- Alojamiento de páginas web para cada proyecto.
- Gestor de proyectos.
- Control gráfico sobre las aportaciones de cada desarrollador en el repositorio.

## Capítulo 4

# Gestión de las imágenes

*El alma nunca piensa sin una imagen mental.*

Aristóteles

**RESUMEN:** Este capítulo describe todos los aspectos relacionados con las imágenes de los documentos. En particular, describe la estructura de directorios que **TEXIS** aconseja, así como los aspectos relacionados con la diferencia entre los formatos esperados cuando se genera el documento final con **latex** y **pdflatex**.

### 4.1. Introducción

En este capítulo tratamos todos los aspectos relacionados con añadir imágenes al documento. Aunque en principio es algo bastante sencillo (desde luego mucho más sencillo que añadir una tabla compleja), existen una serie de cosas a tener en cuenta que merecen un capítulo entero en el manual.

En particular, lo que provoca que las imágenes requieran estas explicaciones detalladas es el hecho de que, como ya dijimos en las secciones ?? y ??, **TEXIS** te permite generar el documento utilizando tanto **latex** como **pdflatex**.

Idealmente, el usuario final de L<sup>A</sup>T<sub>E</sub>X no debería verse influenciado por la aplicación utilizada para generar sus ficheros. Sin embargo, en cierto modo sí se ve afectado; no por el código en sí contenido en los **.tex** sino por los recursos a los que éstos hacen referencia<sup>1</sup>. En concreto, si se utiliza **latex**, las imágenes referenciadas con el comando **\includegraphics** se asume que

---

<sup>1</sup>En ciertas ocasiones también puede verse afectado el código, si se utilizan paquetes que únicamente funcionan con una de ellas.

tienen formato `.eps`, mientras que en cuanto se utiliza `pdflatex`, se admiten `.pdf`, `.png` y `.jpg` (pero no `.eps`).

Por lo tanto, cuando utilizamos un código como el siguiente (similar al que hay en la portada para que aparezca el escudo):

```
\begin{figure}[t]
\begin{center}
\includegraphics[width=0.3\textwidth]%
{Imagenes/Vectorial/escudoUCM}
\caption{Escudo de la Universidad Complutense}
\end{center}
\end{figure}
```

cuando se genera con `latex`, se buscará el fichero `escudoUCM.eps` en el directorio `Imagenes/Vectorial`, mientras que al generarla con `pdflatex`, se buscará el fichero en ese mismo directorio, con el mismo nombre, pero con extensión `.pdf`, `.png` o `.jpg`.

Esto provoca que el programa utilizado para generar el documento final es el que *determina* qué tipo de formato debe usarse para almacenar las imágenes. Existen dos soluciones, que trataremos en las secciones 4.3 y 4.4. Antes de eso, la siguiente sección explica la estructura de directorios que `TEXIS` espera que se utilice.

## 4.2. Gestión de imágenes

Los ficheros de imágenes pueden almacenarse donde el autor del documento desee; al añadir la referencia desde el `.tex`, deberá simplemente indicar la ruta correcta del archivo.

Sin embargo, `TEXIS` propone una estructura determinada que es la que usa este documento. La estructura está elegida de tal forma que facilita la solución del problema de la generación utilizando tanto `latex` como `pdflatex`, por lo que aunque pueda parecer arbitraria, tiene cierto sentido.

La estructura que proponemos empieza con el directorio `./Imagenes`, donde aparecen los siguientes directorios:

- `./Vectorial`: contiene los ficheros correspondientes a imágenes vectoriales.
- `./Bitmap`: contiene los ficheros correspondientes a imágenes de mapas de bits.
- `./Fuentes`: en este directorio aparecen los “fuentes” de las imágenes. Así, si se crean imágenes con Microsoft Visio, Power Point, o Corel, en este directorio irán los ficheros nativos de esos programas. Estos ficheros *no* serán leídos en el proceso de creación del documento final.



Figura 4.1: Figura utilizada para marcar una imagen por hacer.

Cada uno de los directorios anteriores, a su vez, contiene un directorio por capítulo. De esta forma es fácil encontrar los ficheros si se quieren modificar. En el directorio “raíz” se encuentran las imágenes que no pertenecen a ningún capítulo, como la del escudo de la portada. También pueden aparecer otras imágenes que se utilicen en otras partes del documento que no sean los capítulos. Por ejemplo, **TEXIS** proporciona una imagen que puede ser de utilidad, y que está colocada en ese directorio raíz por ser independiente del capítulo. Es una figura “dummy” que sirve para marcar el lugar en el que debería aparecer una figura o gráfico que aún está por hacer (figura 4.1).

**TEXIS** incluye el comando `\figura` para facilitar la inclusión de imágenes. En particular, el comando tiene cuatro parámetros: el nombre del fichero (en el que no hay que indicar el directorio `./Imagenes`), los argumentos pasados al `\includegraphics` que suele tener información sobre el tamaño deseado, la etiqueta con la que luego podrá referenciarse la ilustración, y por último el título que aparecerá en la parte inferior.

Para incluir la figura 4.1 por lo tanto, el comando es<sup>2</sup>:

```
\figura{Vectorial/Todo}{width=.5\textwidth}{fig:todo}%
{Figura utilizada para marcar una imagen por hacer.}
```

que gracias al `{fig:todo}`, luego puede citarse en el código L<sup>A</sup>T<sub>E</sub>X con:

La figura~\ref{fig:todo}  
muestra\ldots

La figura 4.1 muestra...

La figura se añade automáticamente al índice de figuras que aparece al principio del documento, en el que se indica el número de la figura, el texto inferior y la página en la que aparece. Es posible que el texto sea lo suficientemente largo como para que ocupe más de una línea en la entrada en el índice. Si se desea utilizar un texto más corto para evitarlo, se puede

---

<sup>2</sup>Como se ha mencionado, observa que el primer parámetro donde se indica el nombre del fichero *no incluye* ni el nombre del directorio `Imagenes` ni la extensión del fichero.

Programa	Mapa de bits	Vectoriales
<code>latex</code>	<code>.eps</code>	<code>.eps</code>
<code>pdflatex</code>	<code>.png</code>   <code>.jpg</code>	<code>.pdf</code>

Tabla 4.1: Formatos de imágenes para `latex` y `pdflatex`

utilizar el comando `\figuraEx` que recibe un parámetro más con el “título corto” o lo que es lo mismo, con el texto alternativo que aparecerá en el índice.

### 4.3. Formato de las imágenes

Recuperamos en esta sección el problema anteriormente comentado sobre los formatos de las imágenes. Como ya dijimos en la sección de introducción, el uso de `latex` o `pdflatex` determina los formatos de los ficheros que deben utilizarse para las imágenes, según la tabla 4.1.

Esto significa que, en principio, en el momento de añadir la primera imagen al documento, se debe decidir qué programa se utilizará para generarla, y utilizar el formato de imagen adecuado a él. Si tienes claro qué programa utilizarás, la solución es así de simple. Almacena las imágenes en el formato adecuado según la tabla 4.1.

Desgraciadamente, lo habitual es no encontrarse en esa situación. Normalmente cuando se comienza a escribir, es muy difícil pronosticar cuál de los dos se utilizará, y por lo tanto no quieres decantarte por ninguno. No estás seguro de cuálquieres, o puede que quieras poder generarlos de las dos formas, debido a alguna restricción del servicio de publicaciones.

Por lo tanto, la mejor solución es, simplemente, permitir ambas alternativas. Para eso lo más fácil es *duplicar* las imágenes, es decir, mantener tanto la copia que será leída por `latex` como la que utilizará `pdflatex`.

Esta duplicación, no obstante, no suele ser aconsejable, pues (además de consumir más espacio) es propensa a errores: si hay que cambiar una imagen, lo habitual será abrir el fichero de `./Imagenes/Fuentes`, y luego “exportarlo” al formato nativo. En ese momento, es fácil olvidar generar *los dos* ficheros.

Por lo tanto, hay dos soluciones rápidas y fáciles:

- Decidir al principio qué programa se utilizará y utilizar siempre los formatos que éste espera, según la tabla 4.1. Tiene la desventaja de que no se podrá (fácilmente) cambiar el programa generador, pues se necesitará crear las imágenes en el formato esperado por la nueva aplicación.
- No atarse al uso de ninguno de los dos, y duplicar los ficheros de forma que todas las imágenes se guardan dos veces, en cada uno de los forma-

tos esperados por ambos programas. Su desventaja es la duplicación de los ficheros, con los problemas de coherencia que eso puede provocar.

Ninguna de las dos alternativas es óptima, por lo que **TEXIS** proporciona la solución alternativa descrita en la sección siguiente. Si decides utilizar alguna de las opciones fáciles anteriores, puedes omitir la lectura de la misma.

#### 4.4. Imágenes independientes del programa generador

En general conviene evitar duplicar los datos almacenados en disco para evitar problemas de incoherencias. Por eso, cuando no se quiere limitar la generación del documento final a sólo uno de los dos programas, **latex** o **pdflatex**, **TEXIS** desaconseja almacenar en disco cada imagen en los dos formatos exigidos por ellas.

**TEXIS** está preparada para que se almacene en el directorio de las imágenes únicamente las soportadas por **pdflatex** (es decir, ficheros **.pdf** para imágenes vectoriales y **.png** o **.jpg** para mapas de bits). Obviamente, si se hace así, al utilizar **latex** para generar, dará error al no encontrar los ficheros de imágenes correspondientes. Y aquí es donde entra en acción el fichero **Makefile** incluido (que se explica ampliamente en el capítulo 6) cuando se ejecuta con el objetivo **latex**:

```
$ make latex
```

antes de invocar a **latex**, convierte todos los ficheros **.pdf** que hay en el directorio **./Imagenes/Vectorial** a ficheros **.eps**, y todos los **.jpg** y **.png** de **./Imagenes/Bitmap** a **.eps**, para que **latex** los encuentre.

Para realizar la conversión, se utilizan las aplicaciones **pdftops** y **sam2p** que deben estar accesibles en el PATH. Esa es la razón por la que, como mencionábamos en la sección ??, **TEXIS** anima al uso de sistemas Linux: las aplicaciones anteriores están disponibles en este sistema operativo (aunque puede que no se instalen directamente en algunas distribuciones), mientras que en Windows normalmente no están.

#### 4.5. Gestión de imágenes y control de versiones

La solución propuesta en la sección anterior hace que la plantilla espere que dentro del directorio **./Imagenes/Vectorial** aparezcan ficheros **.pdf** y en **./Imagenes/Bitmap** se encuentren ficheros con extensión **.png** o **.jpg**.

En el proceso de generación cuando se utiliza **latex**, se convierten todos esos ficheros a **.eps** para que el programa encuentre las imágenes en el

formato que éste espera, por lo que en los directorios anteriores aparecerán ficheros `.eps` *generados automáticamente*.

`TEXIS` está configurado para que, en caso de utilizar un sistema de control de versiones, éste *ignore* esos ficheros generados (ver una explicación detallada en la sección ??). De esta forma, el usuario no es “molestando” en los momentos de las actualizaciones con mensajes indicando que hay nuevos ficheros en el directorio de las imágenes que no han sido subidos al servidor.

Sin embargo, esta característica debe *anularse* si se utiliza una solución distinta a la indicada en el apartado anterior. En particular, se debe *eliminar* el fichero `.cvsignore`<sup>3</sup> si:

- Se decide al principio de la redacción del documento que se va a utilizar `latex` para su generación (y nunca `pdflatex`), y por lo tanto se usarán siempre ficheros con extensión `.eps`.
- Se decide no utilizar la característica de conversión automática de imágenes, y se duplican los ficheros, guardando siempre tanto la copia leída por `latex` como por `pdflatex`.
- Se decide que las imágenes se guardarán siempre en `.eps` y se cambia el `Makefile` para que, en caso de utilizar `pdflatex` las convierta al formato utilizado por éste (consulta la sección 6.3.1 si estás en este caso).

## 4.6. Imágenes divididas

Somos conscientes de que esta sección incumple lo que comentamos al principio del manual de lo que `TEXIS` *no era*. Decíamos en la sección ?? que esto *no* era un manual de `LATEX`, y lo seguimos manteniendo. Sin embargo, en este apartado incumplimos momentáneamente esa promesa para explicar brevemente cómo incluir varias figuras dentro de un entorno flotante (típicamente otra figura).

En este manual ya ha aparecido un ejemplo. La figura ?? de la página ?? mostraba en realidad dos capturas distintas, cada una de ellas con un subtítulo distinto.

El código `LATEX` de esa figura era:

```
\begin{figure}[t]
  \centering
  %
  \subfloat[] [Propiedades del documento]{
    \includegraphics[width=0.42\textwidth]%
```

---

<sup>3</sup>O no establecer la propiedad `svn:ignore` con él si se utiliza Subversion.

```

    {Imagenes/Bitmap/02/PropiedadesPDF}
    \label{cap2:fig:PropiedadesPDF}
}
\qquad
\subfloat[] [Tabla de contenidos]{
    \includegraphics[width=0.42\textwidth]%
        {Imagenes/Bitmap/02/IndicePDF}
    \label{cap2:fig:TocPDF}
}
\caption{Capturas del visor de PDF\label{cap2:fig:pdf}}
\end{figure}

```

La idea general es crear un entorno figura tradicional, pero no poner en ella directamente el `\includegraphics`, sino subdividir ese entorno figura en varias partes. A cada una de ellas se le da una etiqueta diferente para poder referenciarlas, una descripción, etcétera. Un esquema general sería:

```

\begin{figure}[t]
    \centering
    %
    \subfloat[<ParaElIndice1>] [<Caption1>]{
        % Contenido para este "subelemento" (podrá ser una
        % figura, una tabla, o cualquier otra cosa).
        \includegraphics[width=5cm]{ficheroSinExtension}
        \label{fig:etiqueta1}
    }
    \subfloat[<ParaElIndice2>] [<Caption2>]{
        % Contenido para este "subelemento" (podrá ser una
        % figura, una tabla, o cualquier otra cosa).
        \includegraphics[width=5cm]{ficheroSinExtension}
        \label{fig:etiqueta2}
    }
    \caption{Descripción global para la figura}
    \label{Etiqueta para toda la figura}
\end{figure}

```

El sistema automáticamente decide cuándo poner la siguiente figura al lado, o en otra línea, en base a si entra o no. Es posible forzar a que se ponga en una línea nueva si se deja una línea en blanco en el `.tex`. Esto tiene la repercusión de que no deberías dejar líneas en blanco en ningún momento dentro del entorno flotante, para evitar que se “salte de linea” en las figuras. Si quieras por legibilidad dejar alguna línea, pon un comentario vacío (como hemos hecho con el ejemplo anterior después del `\centering`).

La separación entre dos figuras que se colocan en la misma fila puede ser demasiado pequeña. Para separarlas un poco más, puedes poner `\qquad`

entre el cierre llaves de un `\subfloat` y el siguiente. Ese era el cometido del `\qquad` que aparecía en el ejemplo de las figuras visto anteriormente.

Por otro lado, el `\subfloat` tiene dos “parámetros”, que se colocan entre corchetes justo después. En realidad ambos son opcionales. Podríamos poner directamente:

```
\subfloat{ <comandos para el subelemento> }
```

pero en ese caso no se etiquetará con una letra.

El texto que se pone entre los primeros corchetes se utiliza para el índice de figuras. En teoría, se mostrará en dicho índice primero la descripción global de la figura, y luego la de cada subelemento. Si no quieras que ocurra, deja en blanco el contenido del primer corchete. En la práctica, el índice de figuras no tiene “niveles”, por lo que si se pone la descripción de la subfigura, ésta no aparecerá en el mismo.

El segundo corchete recibe el texto con la descripción del subelemento, es decir lo que aparecerá junto a la letra identificativa. Si lo dejas vacío (pero poniendo los corchetes), saldrá la letra, sin texto. Si ni siquiera pones los corchetes, no saldrá tampoco la letra.

Por último, debes saber que puedes referenciar de forma independiente cada uno de los subelementos. Para eso, basta con `\ref{etiqueta}`, siendo la etiqueta una definida mediante `\label` dentro del `\subfloat`. Se puede entonces referenciar utilizando el comando `\ref` tradicional (que hará que aparezca la letra correspondiente junto con el número de figura), o el comando `\subref`:

```
La figura~\ref{cap2:fig:pdf}
tiene dos partes. La
parte izquierda es la figura~
\ref{cap2:fig:PropiedadesPDF}
y a la derecha está la
\subref{cap2:fig:TocPDF}.
```

La figura ?? tiene dos partes. La parte izquierda es la figura ?? y a la derecha está la ??.

Como ya hemos dicho, se pueden poner varias “filas” de imágenes en la misma figura. Se puede forzar este comportamiento añadiendo líneas en blanco dentro del entorno `\figure` (lo que será interpretado por L<sup>A</sup>T<sub>E</sub>X como un nuevo párrafo), aunque también se añadirán automáticamente si L<sup>A</sup>T<sub>E</sub>X detecta que no entra todo en una única línea. La figura 4.2 (extraída de ?) es un ejemplo de esto. Como se ve en su código L<sup>A</sup>T<sub>E</sub>X que se muestra a continuación, cada una de las imágenes ocupa el 45 % del ancho de la página, por lo que únicamente entran dos figuras por fila. A pesar de que no existe ninguna línea en blanco en el código, las imágenes se colocan en dos filas distintas.

```
\begin{figure}[t]
```



(a) Estudiante y Javy dirigiéndose al barrio de clases



(b) Estudiante enfrente de Framauro



(c) Estudiante interactuando con la pila de operandos



(d) Estudiante hablando con Javy

Figura 4.2: Ejemplo de uso de `subfloat`.

```
\centering
%
\begin{SubFloat}
{\label{fig:cap4:barrioclasses}}
Estudiante y Javy dirigiéndose al barrio de clases
\includegraphics[width=0.45\textwidth]{Imagenes/Bitmap/04/Javy1BarrioClases}
\end{SubFloat}
\qquad
\begin{SubFloat}
{\label{fig:cap4:framauro}}
Estudiante enfrente de Framauro
\includegraphics[width=0.45\textwidth]{Imagenes/Bitmap/04/Javy1Framauro}
\end{SubFloat}
% La siguiente no entra; ira en otra 'linea'
```

```
\begin{SubFloat}
  {\label{fig:cap4:pilaops}%
    Estudiante interactuando con la pila de operandos}%
  \includegraphics[width=0.45\textwidth]%
    {Imagenes/Bitmap/04/Javy1PilaOperandos}%
\end{SubFloat}
\quad
\begin{SubFloat}
  {\label{fig:cap4:estudianteyjavy}%
    Estudiante hablando con Javy}%
  \includegraphics[width=0.45\textwidth]%
    {Imagenes/Bitmap/04/Javy1Javy}%
\end{SubFloat}
\caption{Ejemplo de uso de \texttt{\subfloat}.}%
\label{fig:cap4:javy1}
\end{figure}
```

## Notas bibliográficas

En este capítulo hemos descrito cómo se gestionan las imágenes en **TEX<sup>1</sup>S** por lo que no existe ninguna fuente relacionada adicional de consulta.

Conviene, eso sí, indicar que las explicaciones al respecto del entorno **\subfloat** dadas en la sección 4.6 distan mucho de estar completas, aunque es cierto que deberían ser suficientes para la mayoría de los casos. El entorno, no obstante, permite hacer muchas más cosas. Se puede consultar el manual oficial para más información<sup>4</sup>.

## En el próximo capítulo...

El próximo capítulo pasa a describir algunos aspectos sobre la bibliografía. En concreto, veremos que **TEX<sup>1</sup>S** define un estilo de bibliografía propio que, si bien no introduce demasiadas diferencias con respecto al habitual, permite añadir algún campo nuevo a las citas, como por ejemplo la dirección Web y la fecha de la última vez que se visitó (o comprobó su existencia).

---

<sup>4</sup>Disponible en <ftp://tug.ctan.org/pub/tex-archive/macros/latex/contrib/subfig/subfig.pdf>.

## Capítulo 5

# Bibliografía y acrónimos

*Como un ganso desplumado y escuálido,  
me preguntaba a mí mismo con voz  
indecisa  
si de todo lo que estaba leyendo  
haría el menor uso alguna vez en la vida.*

James Clerk Maxwell, sobre su  
educación en Cambridge

**RESUMEN:** Este capítulo aclara algunas cosas sobre la bibliografía utilizada en TEXIS, y sobre la infraestructura para la creación de una lista de acrónimos.

### 5.1. Bibliografía

Para hacer la bibliografía del documento, TEXIS hace uso, como no podía ser de otra forma, de BibTEX. Esto permite una generación bastante sencilla de la misma, utilizando las entradas `@bibitem` correspondientes. El fichero `makefile` explicado en el capítulo siguiente se encarga de invocar a `bibtex`, la aplicación responsable de la correcta creación de la lista de referencias. Si utilizas cualquier otro sistema para generar el documento final (como por ejemplo las proporcionadas por el editor LATEX que estés utilizando), deberás encargarte de averiguar cómo debes hacerlo.

La sección siguiente localiza el fichero más importante para la construcción de la bibliografía, e indica dónde cambiar los ficheros `.bib` utilizados.

Por otro lado, para añadir en el texto las referencias, TEXIS (y este manual) hacen uso del formato utilizado por el paquete `natbib` que, como se

ha podido comprobar a lo largo de estas páginas, se basa en indicar el nombre del autor y el año de la publicación en el propio texto. La sección 5.1.2 explica brevemente las capacidades del paquete.

**T<sub>E</sub>X<sup>I</sup>S** modifica ligeramente el formato de salida de cada una de las referencias para adecuarlas más a nuestros gustos. También hemos añadido algunas capacidades más que pueden añadirse a las mismas. La sección 5.1.3 las describe.

La parte relativa a la bibliografía termina con una breve sección útil si se quiere cambiar el tipo de bibliografía a utilizar.

### 5.1.1. Ficheros involucrados

El fichero responsable de la generación de la bibliografía en **T<sub>E</sub>X<sup>I</sup>S** es **Cascaras/bibliografia.tex**. Es el responsable de crear el capítulo sin numerar, poner la cabecera especial para que en la parte superior de todas sus páginas aparezca el texto “BIBLIOGRAFÍA”, etc. Para eso, al final del fichero aparece la invocación al comando **\makeBib** definido en **TeXis/TeXis\_bib.tex**.

Ya dijimos en la sección ?? que para cambiar la cita célebre que aparece en el capítulo sin numerar de la bibliografía debemos editar el fichero **Cascaras/bibliografia.tex**. En ese mismo fichero es donde se configuran los archivos donde se deben buscar los **@bibitem** que se referencian en el texto. Para hacerlo, se utiliza el comando **\setBibFiles**:

```
\setBibFiles{%
nuestros, latex, otros%
}
```

Como se puede ver, en este manual las referencias están organizadas en tres ficheros distintos: referencias a trabajos propios (**nuestros.bib**), referencias relacionadas con **L<sup>A</sup>T<sub>E</sub>X** (**latex.bib**) y referencias varias que no entran en ninguna de las dos anteriores (**otros.bib**).

Esos tres ficheros aparecen en el directorio raíz del manual y pueden/deben ser sustituidos por los utilizados en el nuevo documento.

### 5.1.2. Referencias con natbib

Aunque en este manual no se haya hecho un uso demasiado extenso de la bibliografía, es cierto que **natbib** proporciona opciones muy interesantes para utilizarse en textos donde las referencias tengan un peso importante (o al menos más importante que en este manual).

Cuando se utiliza **natbib**<sup>1</sup> en vez de hacer uso de **\cite**, se pueden utilizar otras dos versiones distintas del comando, en concreto **\citet** y

---

<sup>1</sup>El paquete es incluido por **T<sub>E</sub>X<sup>I</sup>S** en **TeXis/TeXis\_pream.tex**.

Comando	Resultado
\citet{key}	Jones et al. (1990)
\citet*{key}	Jones, Baker y Smith (1990)
\citemp{key}	(Jones et al., 1990)
\citemp*{key}	(Jones, Baker y Smith, 1990)
\citemp[cap. 2]{key}	(Jones et al., 1990, cap. 2)
\citemp[e.g.][]{key}	(e.g. Jones et al., 1990)
\citemp[e.g.][p. 32]{key}	(e.g. Jones et al., p. 32)
\citeauthor{key}	Jones et al.
\citeauthor*{key}	Jones, Baker y Smith
\citeyear{key}	1990

Tabla 5.1: Distintas opciones de referencias con `natbib`

\citemp. El primero está pensado para hacer referencias en el propio texto y el último para que las referencias aparezcan entre paréntesis.

En vez de hacer una descripción de cada una de las variaciones, la tabla 5.1 contiene las distintas posibilidades. Las opciones están cogidas directamente de la documentación del paquete y asume que existe un `@bibitem` con nombre `key` que describe una referencia escrita por los autores Jones, Baker y Smith en 1990. Notar que el resultado contiene la conjunción española “y” en vez de la inglesa “and” para separar el último autor.

### 5.1.3. Modificaciones en los `@bibitem`

En TEXIS hemos definido nuestro propio estilo de bibliografía, es decir, el formato de salida de las referencias en el listado que aparece al final del documento. No es demasiado distinto del estilo utilizado por defecto, pero sí tiene ligeras modificaciones.

Las modificaciones más obvias es que se utiliza la versión española del formato, para que aparezca “editor”, “páginas” o “Informe Técnico” entre otros.

También hemos añadido “constantes” (o, en nomenclatura de BibTEX, “macros”) para una editorial y dos series muy utilizadas en nuestra área, “Springer-Verlag” (SV), “Lecture Notes in Computer Science” (LNCS) y “Lecture Notes in Artificial Intelligence (subserie de LNCS)” (LNAI). De esta forma, una entrada de la bibliografía puede ser:

```
@inproceedings{Ejemplo,
  author = { ... },
  title = { ... },
  ...
  publisher = SV,
```

```

series = LNCS,
...
year = 2009,
month = jan
}

```

Como se puede ver se han hecho uso de las macros para indicar la editorial y la serie. También se ha utilizado la macro para indicar el mes (`jan` equivale a enero), de forma que la entrada sea independiente del lenguaje utilizado posteriormente en la referencia.

Mucho más interesante es la creación de *dos nuevos campos* que pueden añadirse en las entradas de BibTEX, y que permiten indicar la página Web en la que se puede encontrar la referencia.

En concreto, admitimos un nuevo campo, `webpage` que permite indicar una página Web, y un campo `lastaccess` permite indicar la fecha del último acceso. De esta forma, la referencia, además de los campos e información habituales, incluye la localización.

Por ejemplo, el libro del Subversion citado en un capítulo anterior (?) está disponible en la Web; para que la URL o dirección Web aparezca en la referencia, se pueden utilizar los nuevos campos:

```

@Book{Subversion,
author      = {Ben Collins-Sussman and
                 Brian W. Fitzpatrick and
                 C. Michael Pilato},
title       = {Version Control with Subversion},
publisher   = {O'Reilly},
year        = 2004,
isbn        = {0-596-00448-6},
webpage     = {http://svnbook.red-bean.com/},
lastaccess  = {Octubre, 2009}
}

```

El estilo añade la cadena “Disponible en” antes de la url (que se añade con el comando `\url` para su correcta división en líneas) y entre paréntesis incluye la fecha del último acceso. El aspecto final de la referencia del libro anterior es:

COLLINS-SUSSMAN, B., FITZPATRICK, B. W. y PILATO, C. M. *Version Control with Subversion*. O'Reilly, 2004. ISBN 0-596-00448-6. Disponible en <http://svnbook.red-bean.com/> (último acceso, Octubre, 2009).

También hemos añadido un nuevo tipo de entrada para referenciar artículos de la Wikipedia. Puede ser discutible si es adecuado o no utilizar citas de la Wikipedia en documentos académicos como tesis o trabajos de fin de

master, pero si eres de los que opinan que debe citarse todo lo que uno utilice y utilizas la Wikipedia, puede que quieras usar esta nueva entrada que hay en TeXIS. Por ejemplo, la entrada de LATEX de la Wikipedia (?) se define con:

```
@Wikipedia{LaTeXWikipedia,
    author      = {Wikipedia},
    title       = {{\LaTeX}},
    wptentry    = {\LaTeX},
    language    = {es},
    webpage     = {http://es.wikipedia.org/wiki/LaTeX},
    lastaccess  = {Mayo, 2009},
}
```

En el texto la referencia no aparece con el año sino que hace alusión a que es una entrada de la Wikipedia, como se ha podido ver en el parrafo anterior. El resultado final en la lista de referencias es:

WIKIPEDIA (LaTeX). Entrada: “LaTeX”. Disponible en <http://es.wikipedia.org/wiki/LaTeX> (último acceso, Mayo, 2009).

#### 5.1.4. Cambio del estilo de la bibliografía

El estilo de las referencias de TeXIS está definido en `TeXiS/TeXiS.bst`, que se incluye en la parte final del fichero `TeXiS_bib.tex`:

```
...
\bibliographystyle{TeXiS/TeXiS}
...
```

Si quieras utilizar un estilo distinto de los disponibles en LATEX basta con que cambies esa línea para definir ese otro estilo. Por ejemplo:

```
...
\bibliographystyle{abbrv}
...
```

configura la bibliografía para que queden numeradas y se referencien desde el texto con el número entre paréntesis<sup>2</sup>. Ten en cuenta, no obstante, que en ese caso el resultado de los comandos `\citep` y `\citet` es el mismo, por lo que puede que necesites reescribir algunas frases; además, `\citeyear` y `\citeauthor` dejarán de funcionar. Por último, cambiar el estilo a un estilo predeterminado como ese elimina la posibilidad de incluir referencias a páginas Web y a la Wikipedia explicadas antes.

---

<sup>2</sup>Si quieras que queden entre corchetes ('[', ']'), debes quitar el `[round]` que aparece en la inclusión del paquete `natbib` en el fichero `TeXiS/TeXiS_pream.tex`.

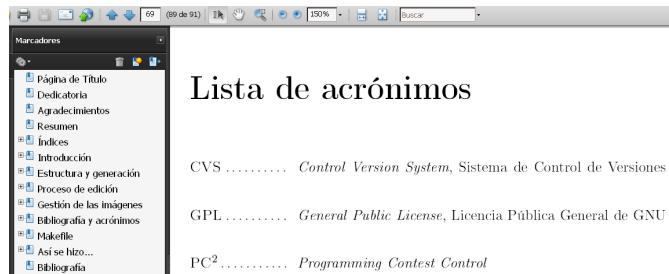


Figura 5.1: Resultado de la lista de acrónimos

Otra posibilidad es que deseas ajustar el funcionamiento del estilo proporcionado por **TEXIS**. En ese caso, puedes editar el fichero del estilo (el antes nombrado **TeXIS/TeXIS.bst**), respetando la sintaxis que espera **bibtex**.

## 5.2. Acrónimos

Los acrónimos son las *siglas* utilizadas a lo largo del documento. **LATEX** permite facilitar su gestión, de manera que se controle automáticamente el momento de la primera aparición de un acrónimo para poner su significado, o para que se genere automáticamente una lista de acrónimos a modo de resumen para ser añadida al final del documento.

Dado que el uso de acrónimos no es tan conocido, de nuevo romperemos nuestra promesa de no explicar aquí aspectos concretos de **LATEX**, y describiremos en la siguiente sección el funcionamiento del paquete **GlossTEX**, que se encarga de la gestión de acrónimos. Despues, nos centraremos en el modo en el que **TEXIS** integra su uso.

### 5.2.1. Acrónimos con glosstex

Al igual que ocurre con la bibliografía, el uso de acrónimos en el documento supone dos cosas:

- A lo largo del texto habrá que indicar en qué punto se usa un acrónimo, al igual que se indica cuando se utiliza una referencia bibliográfica.
- Al final del texto, tendrá que aparecer una lista con todos los acrónimos utilizados (figura 5.1), al igual que ocurre con las citas.

Dada esta similitud con la bibliografía, no sorprende que para que sea **LATEX** quién nos gestione nuestros acrónimos tendremos que hacer uso de una “base de datos” de acrónimos, generalmente en ficheros con extensión **.gdf** (*Glossary Data File*), que son conceptualmente similares a los ficheros **.bib**

usados por BibTEX. Como ejemplo, a continuación se indica el contenido (parcial) del fichero usado en este manual:

```
@entry{CVS, , \emph{Control Version System}, Sistema de Control  
de Versiones}
```

```
@entry{GPL, , \emph{General Public License}, Licencia Pública  
General de GNU}
```

Cada entrada se coloca dentro de un “comando” `@entry`, que tiene tres parámetros, separados por comas:

1. *Acrónimo*: en el ejemplo, CVS y GPL. El acrónimo hace también las veces de etiqueta identificativa, para referenciarlo dentro del texto.
2. *Representación corta*: no aparece en ninguno de los dos ejemplos anteriores (se ha dejado en blanco). Es necesaria únicamente si el acrónimo *tiene formato*. Veremos un ejemplo en un instante.
3. *Versión larga*: contiene la descripción completa del acrónimo. Aunque estemos utilizando comas como separadores de cada uno de los tres elementos, al ser éste el último campo, podría contener comas tal y como muestran los ejemplos.

Una vez que se ha poblado el fichero con los acrónimos, a lo largo del texto es posible añadir comandos de GlossTEX para referirse a ellos, algo conceptualmente similar al uso de `\cite` respecto a la bibliografía. A continuación se muestran los comandos disponibles<sup>3</sup>, y su resultado en el documento final<sup>4</sup>:

- `\acs{CVS}` (*short*): CVS
- `\acl{CVS}` (*long*): CVS
- `\acf{CVS}` (*full*): CVS
- `\ac{CVS}`: es la más interesante de todas. Funciona como `\acf` la primera vez que aparece en el texto, y como `\acs` el resto de las veces.

Las tres primeras muestran diferentes partes de la entrada definida en el fichero `.gdf`. La más completa es la mostrada por `\acf`, que combina la versión corta con la larga, que coloca entre paréntesis. No obstante, el comando más interesante es `\ac`, que “recuerda” si ya se introdujo un acrónimo anteriormente en el documento, mostrando su descripción completa únicamente

---

<sup>3</sup>Para poder usarlos, será necesario haber incluido el paquete `glosstex` en el preámbulo del documento.

<sup>4</sup>Dado que *no* estás compilando el documento con el glosario, *no* podrás ver el resultado correcto aquí. Consulta la sección 5.2.2 para más información.

la primera vez<sup>5</sup>. El modo en el que se construye esta descripción (con la versión larga entre paréntesis) es personalizable, si bien el modo de hacerlo queda fuera del alcance de este documento.

Por tanto, el modo de uso recomendado de los acrónimos es añadir en el fichero `.gdf` todos los acrónimos usados en el documento, y luego hacer *siempre* referencia a ellos a través del comando `\ac`. LATEX incluirá la descripción completa la primera vez, y dejará únicamente el acrónimo todas las demás.

Hemos visto que en los ejemplos descritos, el propio acrónimo *hace las veces de identificador*, de modo que para referenciarlo basta con un mero `\ac{CVS}` o similar. Esto será suficiente la mayor parte de las veces, pero en ocasiones tendremos acrónimos más complejos, como por ejemplo PC<sup>2</sup> (*Programming Contest Control*). El problema, es que para que se muestre correctamente, el código LATEX del acrónimo es en realidad `PC$^2$`, por lo que tendríamos que referirnos a él como `\ac{PC$^2$}`. Esto no sólo resulta incómodo, sino que además genera un error de compilación, dado que no podemos añadir comandos LATEX dentro de un identificador.

La solución es hacer uso del segundo campo que dejábamos vacío en las entradas del fichero `.gdf` de ejemplo:

```
@entry{PC2, PC$^2$, \emph{Programming Contest Control}}
```

Cuando se indica dicho campo, GlossTEX hará uso de él para mostrar el acrónimo, y utilizará el primero sólo como identificador. Ahora, para referirnos a él bastará con un `\ac{PC2}` que, la primera vez que se usa, se convierte en un PC2, y las siguientes en un más corto PC2<sup>6</sup>.

Aparte de la ayuda proporcionada por GlossTEX para evitarnos tener que escribir la versión completa de nuestros acrónimos, también nos sirve para añadir en la parte final un listado de acrónimos donde se muestra el acrónimo y su descripción larga (tal y como mostraba la figura 5.1). Al igual que ocurre con la bibliografía, sólo aparecerán en la lista aquellos acrónimos que realmente se hayan usado a lo largo del documento. El comando para conseguirlo es:

```
\printglosstex(acr)
```

No obstante, y al igual que ocurre con la bibliografía, para que toda esta infraestructura funcione es necesario ejecutar programas externos (aparte del propio `latex` o `pdflatex`). El proceso completo es el siguiente:

---

<sup>5</sup>El uso manual de `acf` *no* hace que GlossTEX considere que el acrónimo ya se ha “presentado”, por lo que si se utiliza primero `acf` y más adelante `ac`, aparecerá de nuevo la versión completa.

<sup>6</sup>De nuevo, como *no* estás compilando el documento con el glosario, *no* podrás ver el resultado correcto aquí.

1. El uso de comandos `\ac?` genera la inclusión de anotaciones dentro de los ficheros `.aux` generados al compilar los `.tex`.
2. Al igual que se hace con la aplicación `bibtex` para la bibliografía, en este caso usamos el programa `glosstex`<sup>7</sup> que los recorre y genera un nuevo fichero con extensión `.gxs` con información sobre las entradas referenciadas, y un `.gxg` con información de registro (*log*).
3. El fichero `.gxs` debe volverse a procesar usando otro programa, en este caso `makeindex`, que ordena alfabéticamente las entradas utilizadas, y les aplica un formato concreto. Este proceso genera un nuevo fichero, con extensión `.glx`, que, finalmente, contiene todo correcto.
4. El fichero anterior es utilizado por el comando `LATEX printglosstex` que debe ser incluido dentro de algún fichero `.tex` del documento, y que es conceptualmente similar al `\bibliography{ficheros}` usado para Bib`TEX`.

Todo esto complica el proceso de construcción del documento, que ahora requiere pasos adicionales<sup>8</sup>:

```
$ pdflatex Tesis
$ bibtex Tesis
$ glosstex Tesis acronimos.gdf
$ makeindex Tesis.gxs -o Tesis.glx -s glosstex.ist
$ pdflatex Tesis
```

En la ejecución de `glosstex` es necesario proporcionarle el nombre del fichero (o ficheros) con la “base de datos” de acrónimos. En el caso de `bibtex` esto no es necesario, porque ya se lo proporcionamos directamente en los `.tex` con el comando `\bibliography{ficheros}`. Además, en la ejecución a `makeindex` proporcionamos el parámetro `glosstex.ist` que indica el formato que queremos aplicar a nuestra lista de acrónimos; ten en cuenta que `makeindex` se utiliza para la generación de otros índices, por lo que para mantener la generalidad mantiene fuera dicho formato para que pueda ser adaptado a cada caso.

Para poder hacer uso, por tanto, de los acrónimos, *es necesario* disponer no sólo del paquete `LATEX glosstex`, sino también de las *aplicaciones glosstex* y `makeindex`. En la sección ?? veíamos un modo simplificado de compilar el presente documento que no incluía las órdenes para procesar los acrónimos. Si no se ejecutan estas aplicaciones, el documento se generará correctamente, salvo por la lista de acrónimos que aparecerá vacía.

---

<sup>7</sup>Este programa deberá estar instalado en el sistema en el que se esté generando el documento.

<sup>8</sup>Como siempre, también es válido el uso de `latex` en lugar de `pdflatex`, pero el fichero generado (`.dvi`) deberá después ser convertido a PDF.

### 5.2.2. Acrónimos en **TEXIS**

Afortunadamente, al utilizar **TEXIS** prácticamente de lo único que hay que preocuparse es de crear la “base de datos” de acrónimos y de hacer uso del comando `\ac` cuando corresponda. El resto de tareas son gestionadas automáticamente<sup>9</sup>.

Para dar soporte al uso de acrónimos, **TEXIS** se apoya en varios ficheros:

- **TeXIS/TeXIS\_acron.tex**: contiene el comando de **GlossTeX** que añade al documento un nuevo capítulo sin numeración con la lista de acrónimos. De manera predefinida, este capítulo se añadirá al documento *unicamente en modo “release”* (consulta la sección ?? para más información). En modo borrador (*debug*) los acrónimos no se incluyen en el documento, de modo que se ahorra algo de tiempo.
- **acronimos.gdf**: éste es el fichero donde se recomienda añadir los acrónimos que se usen. Hace las veces de “base de datos” de acrónimos. Siendo realistas, los ficheros **.tex** de **TEXIS** *no* dependen de que se utilice este fichero en concreto. Tal y como se explicó en la sección anterior, es la ejecución externa de las herramientas de **GlossTeX** la que recibirá el nombre en concreto. Sin embargo, **TEXIS** proporciona un modo automático de construcción, descrito en el capítulo 6, que sí asume este nombre de fichero. La construcción se apoya en la herramienta **make**, de ahí que al principio del documento mencionáramos que resultaba más cómodo hacer uso de plataformas GNU/Linux (que disponen de él) en lugar de Windows.

En la versión en borrador, además de no generarse la lista de acrónimos como un capítulo más, *tampoco* se añaden las descripciones largas de los acrónimos. Es decir, si se utiliza en algún lugar del documento las órdenes de **GlossTeX** mencionadas previamente (`\ac`, `\acl`, etcétera), éstas se “puentearán” de modo que en el documento final tan sólo aparecerá la propia etiqueta. Por ejemplo, dado que esta versión se ha compilado *sin* acrónimos, aparecerá tan sólo CVS aunque en el código **.tex** subyacente haya sido escrito `\acl{CVS}` (puedes comprobarlo si te parece).

Dado que no todos los documentos harán uso de acrónimos, y que, después de todo, generarlos supone un esfuerzo en la fase de compilación no despreciable<sup>10</sup>, es posible que en ocasiones no se deseé que se incluyan los acrónimos tampoco en la versión final (modo *release*). En ese caso, basta con que se modifique el fichero **config.tex** (el mismo en el que se escogía qué versión se quería compilar) y comentar la linea siguiente:

---

<sup>9</sup>Salvo, naturalmente, la instalación de las propias aplicaciones **glosstex** y **makeindex** que deberá haber realizado el usuario.

<sup>10</sup>Este esfuerzo es real tan sólo si no se hace uso del **Makefile** proporcionado por **TEXIS**.

---

```
\def\acronimosEnRelease{1}
```

Si se comenta, **T<sub>E</sub>X<sup>I</sup>S** asumirá que no se desean acrónimos tampoco en la versión final, y no se incluirá el capítulo sin numeración. De nuevo, ten en cuenta que en ese caso los comandos de **GlossT<sub>E</sub>X** se puentearán también<sup>11</sup>.

En el improbable caso en el que se quiera hacer uso de los acrónimos para que sea el propio sistema el que se encargue de escribirnos el significado la primera vez que se usan, pero no se quiere que aparezca el listado final de los acrónimos, entonces será necesario modificar directamente el fichero **Tesis.tex**, y dejar de incluir **TeXiS/TeXiS\_acron**, cuya inclusión es ahora mismo condicional en función de si se usan o no los acrónimos.

Por último, hay que tener en cuenta que cuando se modifica el modelo de compilación (por ejemplo, indicando versión final o borrador, o pidiendo que se añadan o quiten los acrónimos en *release*) es necesario *borrar los ficheros intermedios* generados durante la compilación<sup>12</sup>. Si se está generando el documento haciendo uso de la infraestructura proporcionada por **T<sub>E</sub>X<sup>I</sup>S** a través de su **Makefile** (descrita en el capítulo siguiente), bastará con un sencillo:

```
make clean
```

### 5.2.3. Más allá de **T<sub>E</sub>X<sup>I</sup>S**

El nombre **GlossT<sub>E</sub>X** proviene en realidad de *glossary*, por lo que su objetivo inicial era realizar *glosarios*, no meras listas de acrónimos. Para ser honestos, en las entradas de los ficheros **.gdf** podemos añadir una *descripción completa* con una descripción de la entrada:

```
@entry{PC2, PC$^2$, \emph{Programming Context Control}}
Software de gestión utilizado en los concursos de programación
impulsados por ACM, a través del cual los concursantes envían
sus soluciones, y los jueces acceden a ellas y las valoran.
```

Esa descripción *está fuera* de la entrada **@entry**, y resulta útil en el caso de que quisieramos mantener un glosario de palabras. **T<sub>E</sub>X<sup>I</sup>S** *no* proporciona soporte para esto, por lo que si lo quieres utilizar, tendrás que añadir la infraestructura necesaria por tu cuenta.

---

<sup>11</sup>Aunque en este caso no debería ser un problema porque si no se quieren los acrónimos en la versión final será porque no se han usado.

<sup>12</sup>Las razones que ocasionan esto quedan explicadas en la sección 6.3.2.

## Notas bibliográficas

Existen numerosas publicaciones relacionadas con BibTeX; para una descripción de los tipos de entradas que se soportan, etc., se puede consultar [?](#) o [?](#). En esta última referencia también puede encontrarse información sobre las posibilidades del paquete `natbib`.

Por otro lado, es también fácil encontrar información sobre cómo cambiar el estilo utilizado (es decir, lo que hemos hecho en `TEXIS` para añadir el `@lastaccess` o ajustar las cadenas que aparecen). Por ejemplo, una descripción en español es [?](#). También se puede consultar [?](#) o [?](#).

Respecto a `GlossTeX`, la fuente principal de información es la disponible en el catálogo de paquetes de `TEX`. Puedes encontrarla en <http://www.ctan.org/tex-archive/help/Catalogue/entries/glosstex.html>.

## En el próximo capítulo...

Con este capítulo terminan los capítulos más importantes del manual, donde se ha contado lo que se debe saber para utilizar `TEXIS`. El próximo capítulo describe el fichero `Makefile` proporcionado. El fichero permite generar de forma fácil el documento final, utilizando la utilidad `make` disponible en virtualmente todas las plataformas. Somos conscientes de que no todo el mundo querrá utilizar este mecanismo para generar el documento final (muchos usuarios preferirán utilizar las opciones del editor que utilicen); por eso lo hemos puesto al final.

# Capítulo 6

## Makefile

*A fuerza de construir bien, se llega a  
buen arquitecto.*

Aristóteles

**RESUMEN:** Este capítulo describe la infraestructura de creación del documento final, apoyada en la herramienta `make` de GNU.

### 6.1. Introducción

Ya se ha esbozado a lo largo de este manual que la generación del documento final requiere varias etapas, algo que es de hecho inherente al propio L<sup>A</sup>T<sub>E</sub>X (o PdfL<sup>A</sup>T<sub>E</sub>X). En la sección ?? vimos que era necesario la invocación a `pdflatex` tres veces, junto con el uso de `bibtex`. En la sección 5.2.1 se añadió la necesidad de invocar a `glosstex` y a `makeindex` para añadir el listado de acrónimos. El resultado es una generación bastante laboriosa que requiere dar varios pasos en un orden concreto.

El mundo del desarrollo del software ha lidiado con un problema similar (más complejo, de hecho) desde hace décadas, y que se ha ido resolviendo con diferentes herramientas, siendo `make` una de las más extendidas. T<sub>E</sub>X<sup>I</sup>S proporciona un fichero `Makefile` para ser utilizado con ella, y simplificar la generación del documento, así como otras labores rutinarias. Dado que la infraestructura `make` no está disponible nativamente en plataformas Windows, sólo podrá utilizarse sobre GNU/Linux y otras variantes de Unix<sup>1</sup>. Incluso aunque se utilizara `nmake`, una herramienta similar a `make` proporcionada

---

<sup>1</sup>Windows dispone de Cygwin que proporciona muchas de las herramientas habituales en Unix. No hemos probado T<sub>E</sub>X<sup>I</sup>S sobre él; si lo haces, ¡no dudes en contarnos la experiencia!

con Visual Studio, el `Makefile` de `TEXIS` no funcionaría dado que hace uso de comandos que sólo están disponibles en Unix/Linux. Es por ello que en el primer capítulo recomendábamos el uso de dicho sistema.

En la sección siguiente, se describen los diferentes *objetivos* que este `Makefile` proporciona. La sección 6.3 describe brevemente el funcionamiento de algunas de sus partes.

## 6.2. Objetivos del Makefile

En la terminología de `make`, un objetivo es un *grupo de tareas* que se ejecutan en conjunto. En el entorno del desarrollo del software, esas tareas suelen estar enfocadas a la compilación del proyecto, aunque también se incluyen objetivos para, por ejemplo, borrar los ficheros intermedios, o instalar el programa recién compilado.

El `Makefile` de `TEXIS` sigue esa misma filosofía, proporcionando los siguientes objetivos:

- **pdflatex**: es el objetivo por defecto. Genera el documento utilizando PdfL<sup>A</sup>T<sub>E</sub>X. Se encarga de generar la bibliografía, los acrónimos, y de compilar varias veces el documento para que se actualicen correctamente las referencias.
- **latex**: genera el documento utilizando L<sup>A</sup>T<sub>E</sub>X, y convierte el `.dvi` resultante en `.pdf`. Tiene en cuenta las necesidades en cuanto al formato de las imágenes descritas en la sección 4.4, por lo que se convierten automáticamente a formato `.eps`.
- **imagenes**: se encarga de convertir las imágenes (tanto vectoriales como de mapas de bits) a formato `.eps`. Normalmente este objetivo no necesitará ser lanzado manualmente nunca; el objetivo `latex` anterior lo hace por nosotros.
- **imagenesvectoriales** e **imagenesbitmap**: convierte o bien las imágenes vectoriales, o bien las de mapas de bits a formato `.eps`. Igual que antes, normalmente no se necesitan invocar manualmente.
- **fast**: genera el documento de manera rápida utilizando PdfL<sup>A</sup>T<sub>E</sub>X. Se limita a generarlo una única vez, sin invocar a BibL<sup>A</sup>T<sub>E</sub>X ni a GlossL<sup>A</sup>T<sub>E</sub>X. Está pensada para la compilación “del día a día” cuando se añade algo de texto y se quiere ver rápidamente el resultado, sin preocuparnos de que las referencias queden correctamente actualizadas. Si este objetivo se combina con el comando `\compilaCapítulo` descrito en la sección ??, la compilación puede resultar muy rápida.
- **fastlatex**: similar al anterior, pero realiza la generación utilizando L<sup>A</sup>T<sub>E</sub>X. En este caso, el `.dvi` sigue convirtiéndose a `.pdf`.

- **clean**: elimina todos los ficheros intermedios creados durante la generación del documento. Este objetivo resulta interesante cuando se quiere reconstruir el documento completamente, sin basarse en información previa. Es, de hecho, necesario lanzarlo cuando se compila el documento con la lista de acrónimos por primera vez (o cuando deja de hacerse). Si no se ha modificado la definición de la constante `\acronimosEnRelease` (consulta la sección 5.2.2 para información sobre ella), esto ocurrirá siempre que se cambie el modo de configuración de borrador a versión final (sección ??). Ten en cuenta que este objetivo *borra también* los ficheros `.eps`, por lo que si has modificado el modelo de gestión de imágenes (para que los originales sean `.eps` que se convierten a `.pdf` si se usa PdfL<sup>A</sup>T<sub>E</sub>X, consulta la sección 6.3.1 para más información) entonces tendrás que modificar también este objetivo.
- **distclean**: similar a la anterior, pero también borra el fichero `.pdf` generado, y los ficheros de copia de seguridad de los `.tex` creados por los editores de texto más habituales (con extensiones `.tex~` y `.backup`).
- **crearZip**: genera el documento (con PdfL<sup>A</sup>T<sub>E</sub>X) y crea un fichero `.zip` con él y todos los fuentes (incluido imágenes). El fichero es útil para distribuirlo a revisores que tengan intención de cambiar y regenerar el documento.
- **crearVersion**: similar al anterior, pero copia el `.zip` en el subdirectorio `VisionesPrevias` incluyendo en el nombre la fecha y hora actuales. Es útil para realizar copias de seguridad locales o para “congelar versiones” en “hitos” concretos de la escritura.
- **crearBackup**: genera el documento, y hace una copia de *todo* el directorio (incluyendo el subdirectorio `VisionesPrevias` mencionado antes) comprimiéndola en un fichero `.zip` que copia en el *directorio padre* del actual. Está pensado para hacer una copia de seguridad completa que luego sea guardada en algún otro lugar.
- **ayuda** o **help**: muestra una descripción de todos los objetivos anteriores.

En el día a día, los más útiles son `pdflatex`, `fast` y `clean`. Para invocar a cualquiera de ellos en un entorno GNU/Linux donde `make` esté disponible bastará con:

```
$ make <objetivo>
```

En el caso de que se quiera realizar la generación usando PdfL<sup>A</sup>T<sub>E</sub>X, al ser el objetivo por defecto (el primero que aparece en el `Makefile`) no es necesario especificar nada:

```
$ make
pdflatex Tesis
This is pdfTeXk, Version 3.141592-1.40.3 (Web2C 7.5.6)
[...]
```

Si decides que tu herramienta de generación por defecto sea L<sup>A</sup>T<sub>E</sub>X, seguramente quieras colocar el objetivo `latex` delante para convertirlo en el que se ejecute por defecto.

## 6.3. Funcionamiento interno

En principio, el fichero `Makefile` debería funcionar por sí solo si se siguen los convenios de T<sub>E</sub>X<sup>I</sup>S descritos en los capítulos previos, por lo que la sección anterior debería ser suficiente para un usuario normal. Aquí describiremos algunos detalles internos que pueden ser útiles en algunos (idealmente pocos) casos.

### 6.3.1. La compilación de las imágenes

En el capítulo 4 se mencionaba que T<sub>E</sub>X<sup>I</sup>S es capaz de independizarse de la diferencia entre los formatos de imágenes aceptados por PdfL<sup>A</sup>T<sub>E</sub>X y L<sup>A</sup>T<sub>E</sub>X siempre que el autor siga un determinado convenio en el modo de gestionarlas y utilice la infraestructura de generación, es decir el fichero `Makefile` al que se refiere este capítulo.

Dado que nosotros hacemos normalmente uso de PdfL<sup>A</sup>T<sub>E</sub>X, T<sub>E</sub>X<sup>I</sup>S asume que de manera nativa se querrá utilizar éste en lugar de L<sup>A</sup>T<sub>E</sub>X, por lo que muestra una clara tendencia hacia el formato de imágenes que `pdflatex` soporta de manera nativa.

En concreto, el `Makefile` asume que las imágenes se proporcionan en formato `.pdf` para el caso de las vectoriales, y en formato `.jpg` o `.png` para los mapas de bits. Si las imágenes se desarrollaron originalmente con Kivio, Corel, Visio, Gimp o Photoshop, es responsabilidad del propio autor convertirlas a los formatos soportados por PdfL<sup>A</sup>T<sub>E</sub>X. De esa manera, el `Makefile` no necesitará realizar ningún tipo de transformación de imágenes en el objetivo por defecto `pdflatex`.

En el caso de que se desee utilizar L<sup>A</sup>T<sub>E</sub>X, la situación es más complicada. El `Makefile` tendrá que buscar las imágenes (tanto vectoriales como de mapas de bits) y convertirlas a `.eps`. De eso se encargan los objetivos `imagenesbitmap` y `imagenesvectoriales` respectivamente.

Ambos se apoyan en el convenio de directorios propuesto por T<sub>E</sub>X<sup>I</sup>S, en el que se asume que las imágenes vectoriales estarán en el directorio `Imagenes/Vectorial`, y las de mapas de bits en `Imagenes/Bitmap`, con un

subdirectorio dentro por capítulo<sup>2</sup>. El **Makefile** encadena una serie de invocaciones a otros **Makefile** y algunas llamadas a *scripts* del *shell* (**bash**) para realizar la conversión. En última instancia, el responsable de dicha conversión es un *script* llamado **update-eps.sh**, del que, en realidad, existen dos versiones, uno dentro de **Imagenes/Vectorial** y otro en **Imagenes/Bitmap**, específicos para cada uno de los dos tipos de imágenes. Para convertir los **.pdf** vectoriales a **.eps**, se hace uso de la aplicación **pdftops**, que deberá estar instalada. Por su parte, para convertir las imágenes de mapas de bits **.jpg** o **.png** se usa **sam2p**. Los scripts terminan con error (deteniendo por tanto la generación del documento) si estas herramientas no están disponibles. Se han desarrollado de tal manera que el error únicamente se lance si realmente se necesita convertir alguna imagen. Además, se evita regenerar los ficheros **.eps** si los originales (**.pdf**, **.jpg** o **.png**) no han sufrido cambios desde la última generación. Como ya se dijo previamente, se debe tener en cuenta que el **Makefile** considera a esos ficheros **.eps** como *ficheros generados*, por lo que son eliminados por el objetivo **clean**, y se anima a que se configure el control de versiones (CVS, SVN, etcétera) para que *los ignore*.

Si por alguna razón se prefiere L<sup>A</sup>T<sub>E</sub>X a PdfL<sup>A</sup>T<sub>E</sub>X, entonces resultará más cómodo utilizar el formato **.eps** como predefinido, de manera que a partir de las imágenes originales (creadas con cualquier programa de dibujo) se generen los **.eps** en lugar de los **.pdf** mencionados antes. Si, además, se quiere mantener la posibilidad de generar el documento con **pdflatex** (aunque sea a costa de trabajar más convirtiendo las imágenes), deberá adaptarse la infraestructura de generación en varios puntos:

- Modificar los ficheros **update-eps.sh** que se encuentran en los directorios **Imagenes/Bitmap** y **Imagenes/Vectorial** para que conviertan los **.eps** “fuente” en **.pdf**. En este caso no merece la pena convertir a **.jpg** o **.png** las imágenes vectoriales; resulta más cómodo convertir todo a **.pdf**. Para eso, se puede utilizar **epstopdf** (que deberá estar instalado). Una ventaja extra es que ambas versiones de **update-eps.sh** serán iguales, no como en el caso anterior en el que había que diferenciar entre vectoriales y de imágenes de bits.

Por mantener la coherencia, los ficheros deberían renombrarse a algo como **update-pdf.sh**, en cuyo caso habrá que ajustar los *scripts* **updateAll.sh** para modificar su invocación.

- Modificar el fichero **Makefile** principal (en el directorio “raíz” de T<sub>E</sub>X<sup>I</sup>S) en varios puntos:

---

<sup>2</sup>No se soportan subdirectorios adicionales dentro de los directorios de cada capítulo. Esta restricción se debe al modo en el que los *scripts* buscan los ficheros de imágenes que hay que convertir. Consulta cualquiera de los ficheros **updateAll.sh** dentro de **Imagenes/Vectorial** o **Imagenes/Bitmap** para ver los detalles.

- Poner como objetivo por defecto a `latex` en lugar de a `pdflatex`. Esto es una mera cuestión de comodidad, y para llevarlo a cabo basta colocar en primer lugar el objetivo de `latex`.
- Renombrar `fast` a `fastpdflatex` y `fastlatex` a `fast`. De nuevo, esto es una cuestión de comodidad.
- Modificar el guión (*script*) `cleanAll.sh` situado tanto en el directorio `Imagenes/Vectorial` como en `Imagenes/Bitmap`. Ambos son invocados durante la ejecución del objetivo `clean` del `Makefile` principal. En la versión inicial de `TEXIS`, se borran los ficheros `.eps`, dado que son producto de la compilación. Al usar `LATEX`, hay que conservarlos, y borrar los ficheros `.pdf` generados en los directorios de las imágenes.
- Modificar el fichero `.cvsignore` o el equivalente en el sistema de control de versiones que se esté usando (si hay alguno) para que se ignoren los nuevos tipos de ficheros generados (`.pdf`) en lugar de los `.eps` que vienen predefinidos en `TEXIS`. Consulta la sección 4.5 para más información.

### 6.3.2. Makefile, Gloss`TEX`, y cambio de modo de generación

En la sección 5.2.2 se comentaba que al cambiar el modo de generación de documento desde “depuración” (*debug*) a versión final (*release*) o viceversa, era necesario realizar un `make clean` previo debido al uso de `GlossTEX`. Aunque en la práctica es suficiente con recordar hacerlo, en esta sección se explican las causas de esta necesidad.

Como se recordará de la sección 5.2.1, cuando se utilizan acrónimos, tras varias etapas termina consiguiéndose un fichero con extensión `.glx` con la descripción de aquéllos que se usaron. En la siguiente compilación del documento, el comando `\printglosstex` se encargará de recoger el contenido de dicho fichero e incrustarlo en esa posición.

Ten en cuenta que la primera vez que se compila el documento, *no* existirá ningún fichero `.glx`; esto es perfectamente legal, dado que `\printglosstex` no se quejará si el fichero *no* existe. Por desgracia, sí generará un error si existe *pero está vacío*.

Cuando en `TEXIS` el uso de acrónimos está desactivado (lo que ocurre en la generación de depuración), los comandos de `GlossTEX` como `\acs`, `\acl` etcétera se puentean y no se tienen en cuenta, por lo que en los ficheros intermedios no se informará del uso de ningún acrónimo. Sin embargo, durante el proceso de compilación del documento, el `Makefile` insistirá en realizar los pasos necesarios para la generación de acrónimos (es decir, invocar al ejecutable `glosstex` y a `makeindex`). Éstos desembocarán en la creación de un fichero `.glx` vacío, al no haber ningún acrónimo usado. En principio, esto

significa que `\printglosstex` fallará. Afortunadamente, **TEXIS** *no* añade dicho comando **LATEX** al documento cuando los acrónimos están desactivados, lo que evita el problema.

Sin embargo, cuando se activa su generación (al pasar a modo de compilación *release*), **TEXIS** comienza a incluir el comando. En la primera compilación del documento, por tanto, `\printglosstex` se encontrará que el fichero `.glx` está vacío (de la compilación en *debug* anterior), y fallará. Para solucionarlo, basta en realidad con eliminar dicho fichero; sin embargo, resulta mucho más fácil de recordar (y por tanto práctico) limitarse a realizar un `make clean` que borra, entre otros, el `.glx` por nosotros.

Por otro lado, cuando el documento se ha generado correctamente con los acrónimos (normalmente en modo *release*) y se vuelve a generar sin ellos (en modo *debug*), en la primera compilación **LATEX** (o **PdfLATEX**) hará uso tanto de las fuentes del documento como de los ficheros auxiliares (`.aux`) generados en la compilación anterior para acelerar el proceso. En esos ficheros se encuentran entradas relativas a los acrónimos que se utilizaron en la última compilación (en *release*). Al hacer ahora la compilación sin acrónimos, no se habrán incluido los paquetes **LATEX** que comprenden los comandos de **GlossTEX**, por lo que **LATEX** fallará al no comprenderlos. De nuevo, para solucionar este problema es suficiente con borrar los ficheros `.aux` generados en la compilación anterior; sin embargo resulta mucho más simple realizar un `make clean` con la infraestructura de generación proporcionada por **TEXIS**.

## Notas bibliográficas

Sobre la utilidad `make` hay una gran cantidad de información en Internet. Quizá el lugar de referencia es la página oficial del proyecto de GNU (<http://www.gnu.org/software/make/>), aunque contiene mucha más información de la necesaria para comprender el **Makefile** proporcionado con **TEXIS**. Por el mero hecho de proporcionar también una referencia impresa, puede consultarse también ?.



# Parte I

## Apéndices



# Apéndice A

## Así se hizo...

*Pones tu pie en el camino y si no cuidas tus pasos, nunca sabes a donde te pueden llevar.*

John Ronald Reuel Tolkien, El Señor de los Anillos

**RESUMEN:** Este apéndice cuenta algunos aspectos prácticos que nos planteamos en su momento durante la redacción de la tesis (a modo de “así se hizo nuestra tesis”). En realidad no es más que una excusa para que éste manual tenga un apéndice que sirva de ejemplo en la plantilla.

### A.1. Edición

Ya indicamos en la sección ?? (página ??) que **TEXIS** está preparada para integrarse bien con emacs, en particular con el modo **AucTEX**.

Eso era en realidad un síntoma indicativo de que en nuestro trabajo cotidiano utilizamos emacs para editar ficheros **LATEX**. Es cierto que inicialmente utilizamos otros editores creados expresamente para la edición de ficheros en **LATEX**, pero descubrimos emacs y ha llegado para quedarse (la figura ?? mostraba una captura del mismo mientras creábamos este manual). Ten en cuenta que si utilizas Windows, también puedes usar emacs para editar; no lo consideres como algo que sólo se utiliza en el mundo Unix. Nosotros lo usamos a diario tanto en Linux como en Windows.

No obstante, hay que reconocer que emacs *no* es fácil de utilizar al principio (el manual de referencia de ? tiene más de 550 páginas); su curva de aprendizaje es empinada, especialmente si quieres sacarle el máximo partido, o al menos beneficiarte de algunas de sus combinaciones de teclas. Pero una vez que consigues *no* mover las manos para desplazar el cursor sobre

el documento, manejas las teclas rápidas para añadir los comandos L<sup>A</sup>T<sub>E</sub>X más utilizados y conoces las combinaciones de Auc<sup>T</sup>EX para moverte por el documento o buscar las entradas de la bibliografía, no cambiarás fácilmente a otro editor.

Si quieras aprovechar emacs, no debes dejar de leer el documento que nos introdujo a nosotros en el modo Auc<sup>T</sup>EX, “*Creación de ficheros L<sup>A</sup>T<sub>E</sub>X con GNU Emacs*” (?).

## A.2. Encuadernación

Si has mirado con un poco de atención este manual, habrás visto que los márgenes que tiene son bastante grandes. T<sub>E</sub>X<sup>I</sup>S no configura los márgenes a unos valores concretos sino que, directamente, utiliza los que se establecen por defecto en la clase `book` de L<sup>A</sup>T<sub>E</sub>X.

Aunque es más o menos reconocido que si L<sup>A</sup>T<sub>E</sub>X utiliza esos márgenes debe tener una razón de peso (y de hecho la tiene, se utilizan esos para que el número de letras por línea sea el idóneo para su lectura), cuando se comienza a mirar el documento con los ojos del que quiere verlo encuadrado, es cierto que parecen excesivos. Y empiezas a abrir libros, regla en mano, para medir qué márgenes utilizan. Y reconoces que son mucho más pequeños (y razonables) que el de tu maravilloso escrito. Al menos ese fue nuestro caso.

En ese momento, una solución es *reducir* esos márgenes para que aquello quede mejor. Sin embargo nuestra opción no fue esa. Si tu situación te permite *no* encuadrinar el documento en formato DIN-A4, entonces puedes ir a la reprografía de turno y pedir que, una vez impreso, te guillotinen esos márgenes.

Tu escrito quedará entonces en “formato libro”, mucho más manejable que el gran DIN-A4, y con unos márgenes mucho más razonables. La figura A.1 muestra el resultado, comparando el tamaño final con el de un folio, que aparece superpuesto.

## A.3. En el día a día

Para terminar este breve apéndice, describimos ahora un modo de trabajo que, si bien no utilizamos en su día para la escritura de la tesis, sí hemos utilizado desde hace algún tiempo para el resto de nuestros escritos de L<sup>A</sup>T<sub>E</sub>X, incluidos T<sub>E</sub>X<sup>I</sup>S y éste, su manual.

Estamos hablando de lo que se conoce en el mundo de la ingeniería del software como *integración continua* (?). En concreto, la integración continua consiste en aprovecharse del servidor del control de versiones para realizar, en cada *commit* o actualización realizada por los autores, una comprobación de si los ficheros que se han subido son de verdad correctos.

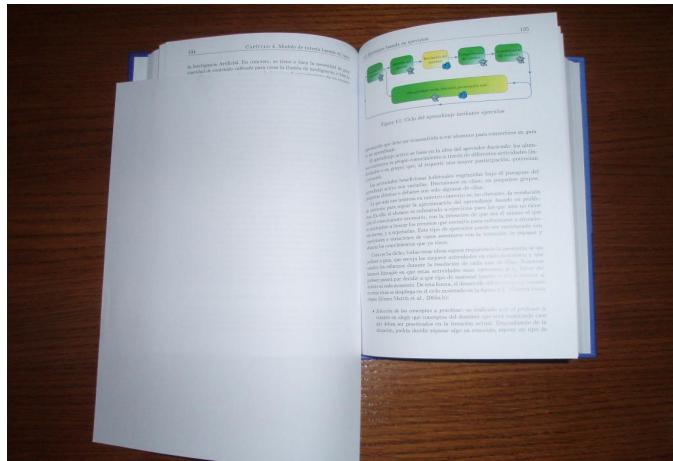


Figura A.1: Encuadernación y márgenes guillotinados

En el mundo del desarrollo software donde un proyecto puede involucrar decenas de personas realizando varias actualizaciones diarias, la integración continua tiene mucha importancia. Después de que un programador realice una actualización, un servidor dedicado comprueba que el proyecto sigue compilando correctamente (e incluso ejecuta los test de unidad asociados). En caso de que la actualización haya estropeado algo, el servidor de integración envía un mensaje de correo electrónico al autor de ese *commit* para avisarle del error y que éste lo subsane lo antes posible, de forma que se perjudique lo menos posible al resto de desarrolladores.

Esa misma idea la hemos utilizado en la elaboración de **TEXIS** y de este manual. Cada vez que uno de los autores subía al SVN algún cambio, el servidor comprobaba que el fichero maestro seguía siendo correcto, es decir, que se podía generar el PDF final sin errores.

No entraremos en más detalles de cómo hacer esto. El lector interesado puede consultar ?. Como se explica en ese artículo algunas ventajas del uso de esta técnica son:

- Se tiene la seguridad de que la versión disponible en el control de versiones es válida, es decir, es capaz de generar sin errores el documento final.
- Se puede configurar el servidor de integración continua para que cada vez que se realiza un *commit*, envíe un mensaje de correo electrónico a todos los autores del mismo. De esta forma todos los colaboradores están al tanto del progreso del mismo.
- Se puede configurar para que el servidor haga público (via servidor Web) el PDF del documento (ver figura A.2a). Esto es especialmente

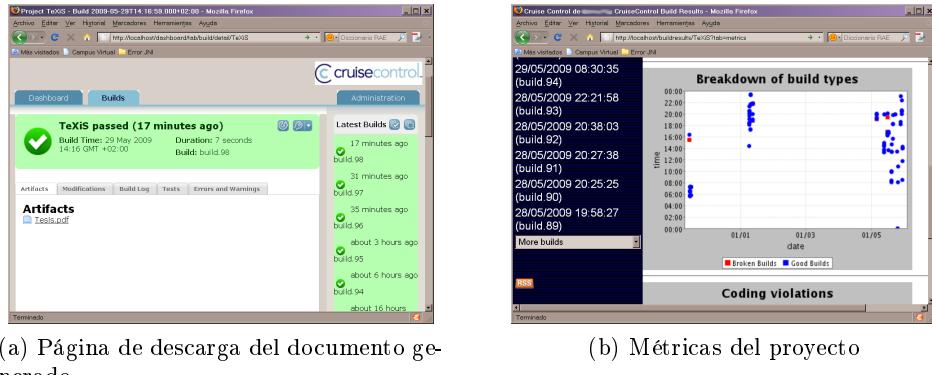


Figura A.2: Servidor de integración continua

útil para revisores del texto como tutores de tesis, que no tendrán que preocuparse de descargar y compilar los `.tex`.

Por último, el servidor también permite ver la evolución del proyecto. La figura A.2b muestra una gráfica que el servidor de integración continua muestra donde se puede ver la fecha (eje horizontal) y hora (eje vertical) de cada *commit* en el servidor; los puntos rojos representan commits cuya compilación falló.

*-¿Qué te parece desto, Sancho? – Dijo Don Quijote –  
Bien podrán los encantadores quitarme la ventura,  
pero el esfuerzo y el ánimo, será imposible.*

*Segunda parte del Ingenioso Caballero  
Don Quijote de la Mancha  
Miguel de Cervantes*

*-Buena está – dijo Sancho –; fírmela vuestra merced.  
–No es menester firmarla – dijo Don Quijote–,  
sino solamente poner mi rúbrica.*

*Primera parte del Ingenioso Caballero  
Don Quijote de la Mancha  
Miguel de Cervantes*

