Mi tesis doctoral

Perico de los Palotes

29 de enero de 2008

Introducción

Con este ejemplo queremos mostrar la forma en que se estructura una tesis con LaTeX. En la primera parte esencialmente sólo hay un poco de lo que sería el «esqueleto» de una tesis, con capítulos secciones e índices. En la segunda parte hay construcciones concretas de listas de ítems, gráficos, bibliografía, índice terminológico, citas textuales sacadas de otro lugar, etc.

En ocasiones los textos servirán para explicar cosas, al tiempo que se realizan. Pero en otras puede tratarse unicamente de textos de relleno cuyo contenido carece de interés en este contexto.

Índice general

Índice de figuras

Índice de cuadros

Parte I

Marco teórico

Capítulo 1

La prehistoria

1.1. En Ática

1.2. En Asia

Capítulo 2

La Edad Antigua

2.1. En Grecia

2.2. En Roma

Capítulo 3

Software libre

En capítulos no muy largos y cuya estructura véase [4] de secciones está clara, puede hacerse un único archivo. En capítulos complejos con múltiples secciones donde todavía no tengo decidido donde poner cada cosa es aconsejable hacer una archivo para cada sección. De esa manera será muy sencillo poder hacer cambios. En este capítulo vamos a jugar con esa idea. Utilizaremos texto irrelevante tomado de cualquier lugar de internet.[3]

3.1. El Proyecto GNU. Richard Stallman

La primera comunidad que comparte el software

Cuando comencé a trabajar[1] en el Laboratorio de Inteligencia Artificial del MIT en 1971, me incorporé a una comunidad que compartía el software que ya tenía varios años de existencia. El acto de compartir software no estaba limitado a nuestra comunidad en particular; es tan antiguo como las computadoras, de la misma manera que compartir recetas es tan antiguo como cocinar. Pero nosotros lo hacíamos en mayor grado que la mayoría de los otros.

El Laboratorio de IA usaba un sistema operativo denominado ITS (Incompatible Timesharing System) [Sistema incompatible de tiempo compartido] que los hackers (1) del staff habían diseñado y escrito en lenguaje ensamblador para la PDP-10 de Digital, una de las más grandes computadoras de la época. Mi trabajo como miembro de esta comunidad, como hacker de sistema en el staff del laboratorio de IA, era mejorar este sistema.

No denominábamos «software libre» a nuestro software porque dicho término no existía; pero éso es lo que era. Cuando alguien de otra universidad o compañía deseaba portar y usar un programa, lo permitíamos con gusto. Si usted veía a alguien usando un programa interesante y poco conocido, siempre se podía pedir el código fuente para verlo, de manera que uno podía

[1] Este artículo está tomado de www.gnu.org/gnu/thegnuproject.es.html

leerlo, cambiarlo, o canibalizar ciertas partes del mismo para hacer un nuevo programa.

(1) El uso de «hacker» para referirse al «quebrantador de la seguridad» es una confusión proveniente de los medios masivos. Nosotros los hackers nos negamos a reconocer dicho significado, y continuamos utilizando la palabra para indicar a «alguien apasionado por la programación y que disfruta al ser hábil e ingenioso».

El colapso de la comunidad

La situación cambió drásticamente durante la primera parte de los 1980s cuando Digital discontinuó la serie PDP-10. Su arquitectura, elegante y poderosa en los 60s, no se pudo extender naturalmente a los espacios de direccionamiento más grandes que se hicieron factibles en los 80s. Esto significó que prácticamente todos los programas que componían a ITS se volvieron obsoletos.

La comunidad de hackers del laboratorio de IA ya se había colapsado, cierto tiempo antes. En 1981, la compañía derivada Symbolics había contratado a casi todos los hackers del laboratorio de IA, y la despoblada comunidad ya no era capaz de mantenerse a sí misma. (El libro Hackers, de Steve Levy, describe estos eventos, y muestra un claro panorama de esta comunidad en sus comienzos.) Cuando el laboratorio de IA adquiere una nueva PDP-10 en 1982, sus administradores deciden utilizar el sistema no libre de tiempo compartido de Digital en lugar de ITS.

Las computadoras modernas de esa época, como la VAX o el 68020, tienen sus propios sistemas operativos, pero ninguno de ellos es software libre: usted debe firmar un «acuerdo de no revelar» (nondisclosure agreement) aún para obtener una copia ejecutable.

Esto quiere decir que el primer paso para poder utilizar una computadora era prometer que no ayudaría a su vecino. Se prohibía la existencia de una comunidad cooperativa. La regla hecha por los dueños de software privativo era: «si usted comparte con su vecino, usted es un pirata. Si desea algún cambio, ruéguenos para que lo hagamos nosotros».

La idea de que el sistema social del software privativo–el sistema que dice que usted no tiene permitido compartir o cambiar el software– es antisocial, que no es ético, que está sencillamente equivocado, puede ser una sorpresa para algunos lectores. ¿Pero qué otra cosa podríamos decir sobre un sistema que se basa en dividir el público e impide socorrer a los usuarios? Los lectores que se sorprendan por esta idea es porque han tomado el sistema social del software privativo tal como se lo han dado, o porque lo han juzgado en función de los términos sugeridos por las empresas que hacen software privativo. Los publicadores de software han trabajado duro y parejo para convencer a las personas de que solamente hay una manera de ver este tema.

Cuando los publicadores de software habla de «hacer valer» sus «dere-

chos» o de «detener la piratería», lo que *dice* es secundario. El mensaje real de estas declaraciones está en las presunciones no declaradas que ellos dan por sentado; se supone que el público debe aceptarlas de manera acrítica. Así que examinémoslas.

Una de las presunciones es que las compañías de software tienen un derecho natural incuestionable que las habilita para ser dueñas de un software, y por lo tanto a disponer de poder sobre todos los usuarios del mismo. (Si éste fuera un derecho natural, entonces sin importar cuánto daño le causare al público, no podríamos objetarlo.) De manera muy interesante, la Constitución de los Estados Unidos de América y la tradición legal rechazan esta visión; el copyright no es un derecho natural, sino un monopolio artificial impuesto por el gobierno que limita el natural derecho a copia de los usuarios.

Otra presunción no declarada es que la única cosa importante sobre del software es qué trabajo le permite realizar a usted–que a nosotros los usuarios de computadoras no nos debe importar qué clase de sociedad nos permiten tener.

Una tercera presunción es que no tendríamos software utilizable (o, que nunca tendríamos un programa para hacer tal o cual trabajo en particular) si no le ofrecemos a una compañía poder sobre los usuarios de dicho programa. Esta presunción puede haber sonado plausible, antes de que el movimiento por el software libre demostrara que podemos hacer abundante software útil sin ponerle cadenas.

Si nos resistimos a aceptar dichas presunciones, y juzgamos acerca de estos temas sobre la base moral que nos da el sentido común ordinario y ponemos al usuario en primer lugar, arribaremos a conclusiones muy distintas. Los usuarios de computadoras deben tener libertad para modificar los programas para ajustarlos a sus necesidades, y libertad para compartir el software, porque la base de la sociedad está en ayudar a las otras personas.

No se dispone aquí del espacio necesario para explayarnos en el razonamiento que hay detrás de esta conclusión, y por ese motivo pido al lector que vea la página web www.gnu.org/philosophy/why-free.es.html

Una elección moral severa.

Al desaparecer mi comunidad, se hizo imposible continuar como antes. En lugar de ello, me enfrenté a una elección moral severa.

La elección fácil era unirme al mundo del software privativo, firmar los acuerdos de no revelar, y prometer que no iría en ayuda de mi amigo hacker. Es muy probable que desarrollara software que se entregaría bajo acuerdos de no revelar y de esa manera incrementara también las presiones sobre otra gente para que traicionen a sus compañeros.

Podría haber hecho dinero de esta manera, y tal vez me hubiese divertido escribiendo código. Pero sabía que al final de mi carrera, al mirar atrás a los

años construyendo paredes para dividir a la gente, sentiría que usé mi vida para empeorar el mundo.

Ya había estado del lado en que se reciben los acuerdos de no revelar, por experiencia propia, cuando alguien se negó a entregarme, a mí y al Laboratorio de IA del MIT, el código fuente del programa de control de nuestra impresora. (La ausencia de ciertas características en este programa hacía que el uso de la impresora fuera frustrante en extremo.) Así que no podía decirme a mí mismo que los acuerdos de no revelar son inocentes. Me enojó mucho cuando él se negó a compartir con nosotros; no podía ahora cambiarme de lugar y hacerle lo mismo a todos los demás.

Otra elección, fácil pero dolorosa, era abandonar el campo de la computación. De esta manera no se usarían mis habilidades para mal, pero aún así se desperdiciarían. Yo no sería culpable por dividir y restringir a los usuarios de computadoras, pero ello sucedería igual.

Así que busqué la manera en la cual un programador podría hacer algo para bien. Me pregunté: ¿habrá algún programa o programas que yo pueda escribir, de tal manera de otra vez hacer posible una comunidad?

La respuesta era clara: lo primero que se necesitaba era un sistema operativo. Este es el software crucial para empezar a usar una computadora. Con un sistema operativo usted puede hacer muchas cosas; sin uno, ni siquiera puede funcionar la computadora. Con un sistema operativo libre, podríamos tener de nuevo una comunidad de hackers cooperando–e invitar a cualquiera a unírsenos. Y cualquiera sería capaz de utilizar una computadora sin que de movida conspire a favor de la privación de sus amigas o amigos.

Como desarrollador de sistema operativo, tengo las habilidades apropiadas para esa tarea. Así que aún cuando no tenía garantías de éxito, me dí cuenta que había sido elegido para hacer ese trabajo. Decidí hacer que el sistema fuese compatible con Unix pues así sería portable, y los usuarios de Unix podrían cambiarse a él con facilidad. El nombre GNU se eligió siguiendo una tradición hacker, como acrónimo recursivo para «GNU's Not Unix».

Un sistema operativo es más que un núcleo, apenas suficiente para hacer funcionar otros programas. En los 1970s, todo sistema operativo digno de llamarse así incluía procesadores de órdenes, ensambladores, compiladores, intérpretes, depuradores, editores de texto, programas de correo, y muchos otros. ITS los tenía, Multics los tenía, VMS los tenía, Unix los tenía. El sistema operativo GNU también los incluiría.

Más adelante escuché estas palabras, atribuídas a Hillel (1):

Si yo no me preocupo por mí mismo, ¿quién lo hará por mí? Si sólo me preocupo por mí mismo, ¿qué es lo que soy? Si no lo hago ahora, ¿cuándo?

La decisión de iniciar el proyecto GNU se basó en un espíritu similar.

(1) Como ateo que soy, no soy seguidor de ningún líder religioso, pero algunas veces encuentro que admiro alguna cosa que dijo uno de ellos.

Libre como en libertad

El término «free software» [N. del T.: en inglés free = libre o gratis] se malinterpreta a veces–no tiene nada que ver con el precio. El tema es la libertad. Aquí, por lo tanto, está la definición de software libre: un programa es software libre, para usted, un usuario en particular, si:

- Usted tiene libertad para ejecutar el programa, con cualquier propósito.
- Usted tiene la libertad para modificar el programa para adaptarlo a sus necesidades. (Para que esta libertad sea efectiva en la práctica, usted debe tener acceso al código fuente, porque modificar un programa sin disponer del código fuente es extraordinariamente dificultoso.)
- Usted tiene la libertad para redistribuir copias, tanto gratis como por un cánon.
- Usted tiene la libertad para distribuir versiones modificadas del programa, de tal manera que la comunidad pueda beneficiarse con sus mejoras.

Como «free» [libre] se refiere a libertad y no a precio, no existe contradicción entre la venta de copias y el software libre. De hecho, la libertad para vender copias es crucial: las colecciones de software libre que se venden en CD-ROM son importantes para la comunidad, y la venta de las mismas es una manera importante de obtener fondos para el desarrollo de software libre. Por lo tanto, si la gente no puede incluir un programa en dichas colecciones, dicho programa no es software libre.

A causa de la ambigüedad de «free», la gente ha estado buscando alternativas, pero nadie ha encontrado una alternativa apropiada. El idioma inglés tiene más palabras y matices que ningún otro, pero carece de una palabra simple, no ambigüa que signifique «libre», como en libertad–«unfettered» [sin cadenas] es la palabra que más se acerca en significado. Otras alternativas como liberated [liberado], freedom [libertad] y open [abierto] tienen el significado equivocado o alguna otra desventaja.

Software GNU y el sistema GNU

El desarrollo de un sistema complejo es un proyecto de gran envergadura. Para ponerlo dentro de mi alcance, decidí adaptar y usar las piezas existentes de software libre siempre que fuera posible. Por ejemplo, en los mismos comienzos decidí que TeX sería el principal compaginador de texto; unos pocos años más tarde, decidí que usaría el sistema X Window, en lugar de escribir otro sistema de ventanas para GNU.

A causa de esta decisión, el sistema GNU no coincide con la suma de todo el software GNU. El sistema GNU incluye programas que no son software GNU, programas que fueron desarrollados por otras personas y proyectos para sus propios propósitos, pero que nosotros podemos utilizar porque constituyen software libre.

El inicio del proyecto

En enero de 1984 renuncié a mi trabajo en el MIT y comencé a escribir software GNU. Era necesario abandonar el MIT, para que el MIT no interfiriera con la distribución de GNU como software libre. Si hubiese continuado como parte del staff, el MIT podría haber reclamado propiedad sobre el trabajo, y podría haber impuesto sus propios términos de distribución, o incluso podría haberlo transformado en un paquete de software privativo. Yo no tenía la intención de hacer un trabajo enorme sólo para ver que perdía la utilidad para la cual se había realizado: crear una nueva comunidad para compartir software.

Sin embargo, el Profesor Winston, por entonces a cargo del Laboratorio de IA del MIT, me invitó amablemente a que continúe utilizando las instalaciones del Laboratorio.

Los primeros pasos

Poco después de comenzar en el proyecto GNU, escuché acerca del Free University Compiler Kit [Kit de Compilador de la Universidad Libre], también conocido como VUCK. (La palabra Holandesa para free comienza con una V.) Se trataba de un compilador diseñado para manejar múltiples lenguajes, C y Pascal entre ellos, y para admitir múltiples máquinas destino. Le escribí a su autor para consultarle si GNU lo podría usar.

Él me respondió burlonamente, dejando en claro que la universidad era libre, pero el compilador no. Por lo tanto, decidí que mi primer programa para el proyecto GNU sería un compilador multilenguaje, multiplataforma.

Con la esperanza de evitar tener que escribir todo el compilador por mí mismo, obtuve el código fuente del compilador Pastel, que era un compilador multiplataforma desarrollado en el «Lawrence Livermore Lab». Admitía, y estaba escrito en una versión extendida de Pascal, diseñada para usarse como lenguaje de programación a nivel de sistema. Le agregué un front end para C, y comencé a transportarlo a la computadora Motorola 68000. Pero tuve que abandonar la idea al descubrir que el compilador necesitaba varios megabytes de espacio en la pila, y los sistemas Unix basados en 68000 sólo permitían 64 kbytes.

Fue entonces cuando me dí cuenta que el compilador Pastel funcionaba analizando el fichero de entrada completo y transformándolo en un árbol sintáctico, luego convertía todo el árbol sintáctico en una cadena de «instruc-

ciones» y luego generaba el fichero entero de salida, y en ningún momento liberaba el espacio ocupado. En ese momento llegué a la conclusión de que debería escribir un nuevo compilador partiendo desde cero. Ese nuevo compilador se conoce ahora como GCC; no hay nada del compilador Pastel en él, pero me las arreglé para adaptar y usar el front end que había hecho para C. Pero eso pasó unos años más tarde; primero, trabajé sobre GNU Emacs.

GNU Emacs

Comencé a trabajar sobre GNU Emacs en setiembre de 1984, y al principio de 1985 ya empezaba a ser usable. Esto me permitió usar sistemas Unix para las tareas de edición; como no tenía ningún interés en aprender a usar vi o ed, había realizado mis tareas de edición en otras clases de máquinas hasta ese momento.

A estas alturas, la gente comenzó a querer usar Emacs, con lo que apareció el tema de cómo distribuirlo. Por supuesto, lo puse en el servidor de FTP anónimo de la computadora del MIT que usaba. (Esta computadora, prep.ai.mit.edu, se transformó a causa de ello en la sede principal de distribución a través de FTP de GNU; cuando fue decomisada unos años después, transferimos el nombre a nuestro nuevo servidor FTP.) Pero en aquella época, mucha gente interesada no estaba en Internet y no podía obtener una copia por FTP. Así que la pregunta era: ¿qué tendría que decirles a ellos?

Podría haber dicho, «Busque un amigo que esté en la red y que haga una copia para usted». O podría haber hecho lo que hice con el Emacs para PDP-10 original, decirles: «Envíeme por correo una cinta y un sobre con su dirección y los sellos de correo necesarios, y yo le devolveré la cinta con Emacs dentro». Pero no tenía trabajo, y estaba buscando de qué manera podía hacer dinero con el software libre. Entonces anuncié que le enviaría la cinta a quien me la pidiera, mediante el pago de un cánon de $150. De esta manera, inicié un negocio de distribución de software libre, el precursor de las compañías que en la actualidad distribuyen completos sistemas GNU basados en Linux.

¿Es libre el programa para cualquier usuario?

Si un programa es software libre cuando abandona las manos de su autor, esto no significa que será software libre para todos los que tienen una copia de él. Por ejemplo, el software de dominio público (software que no está sujeto al copyright de nadie) es software libre; pero cualquiera puede hacer una versión modificada propietaria a partir de él. En ese mismo sentido, muchos programas libres están sujetos a copyright pero se distribuyen mediante sencillas licencias permisivas que admiten las versiones modificadas propietarias.

El ejemplo paradigmático de este problema es el X Window System.

Desarrollado en el MIT, y entregado como software libre con un licencia permisiva, fue rápidamente adoptado por varias compañías de computación. Éstas agregaron X a sus sistemas Unix privativos, sólo en formato binario, y lo cubrieron con el mismo acuerdo de no revelar. Estas copias de X eran tanto (software) libres en cuanto lo era el Unix.

Los desarrolladores del X Window System no consideraban que esto fuese un problema–esperaban y buscaban que esto sucediese. Su meta no era la libertad, sólo el «éxito», definido como «tener muchos usuarios». No les preocupaba si esos usuarios tenían libertad, sólo que sean numerosos.

Esto nos lleva a una situación paradójica en la cual dos maneras distintas de contabilizar la cantidad de libertad dan por resultado dos respuestas distintas a la pregunta «¿Es libre este programa?». Si usted juzga en base a la libertad que se proporcionaba con los términos de distribución de la entrega del MIT, diría que X es software libre. Pero si usted mide la libertad del usuario promedio de X, diría que X es software privativo. La mayoría de los usuarios de X usan las versiones propietarias que vienen con los sistemas Unix, no la versión libre.

Copyleft y la GNU GPL

La meta de GNU era dar libertad a los usuarios, no sólo ser popular. Por lo tanto, debíamos usar términos de distribución que impidieran que el software GNU se transformara en software privativo. El método que utilizamos se denomina «copyleft».(1)

El copyleft usa la ley de copyright, pero la da vuelta para servir a lo opuesto de su propósito usual: en lugar de ser un medio de privatizar el software, se transforma en un medio de mantener libre al software.

La idea central del copyleft es que le damos a cualquiera el permiso para correr el programa, copiar el programa, modificar el programa y redistribuir versiones modificadas–pero no le damos permiso para agregar restricciones propias. De esta manera, las libertades cruciales que definen al «software libre» quedan garantizadas para cualquiera que tenga una copia; se transforman en derechos inalienables.

Para que el copyleft sea efectivo, las versiones modificadas deben ser también libres. Esto asegura que todo trabajo basado en el nuestro quedará disponible para nuestra comunidad si se publica. Cuando los programadores que tienen trabajo como programadores se ofrecen como voluntarios para mejorar un software GNU, es el copyleft lo que impide que sus empleadores digan: «no puede compartir esos cambios, porque los queremos usar para hacer nuestra versión propietaria del programa».

El requerimiento de que los cambios deben ser libres es esencial si queremos asegurar la libertad para cada usuario del programa. Las compañías que privatizaron el X Window System en general realizaron algunos cambios para transportarlo a sus sistemas y hardware. Estos cambios fueron pequeños

comparados con el gran tamaño de X, pero no fueron triviales. Si el hacer cambios fuera una excusa para negar libertad a los usuarios, sería fácil para cualquiera tomar ventaja de la excusa.

Un tema relacionado trata la combinación de un programa libre con código no libre. Tal combinación será inevitablemente no-libre; cualesquiera libertades que falten a la parte no-libre, le faltarán también al todo. Si se permiten tales combinaciones se abriría un agujero lo suficientemente grande como para hundir el barco. Por ello, un requerimiento crucial para el copyleft es que se tape este hoyo: cualquier cosa agregada a o combinada con un programa bajo copyleft debe ser tal que la versión combinada total sea también libre y bajo copyleft.

La implementación específica de copyleft que usamos para la mayoría del software GNU es la Licencia Pública General de GNU (GNU General Public License) o LPG GNU para abreviar. Tenemos otras clases de copyleft que se usan en circunstancias específicas. Los manuales GNU también están bajo copyleft, pero utilizamos un copyleft mucho más simple, porque no es necesaria la complejidad de la LPG GNU para los manuales.

(1) En 1984 o 1985, Don Hopkins (un compañero muy imaginativo) me envío una carta por correo. En el sobre, escribió varios dichos divertidos, entre ellos éste: «Copyleft–all rights reversed» [Copyleft–todos los derechos reversados"]. Utilicé la palabra «copyleft» para denominar al concepto de distribución que estaba desarrollando en esa época.

La Fundación para el Software Libre

A medida que el interés en el uso de Emacs crecía, otras personas se involucraron en el proyecto GNU, y decicimos que era el momento de buscar fondos nuevamente. Por ello en 1985 creamos la «Free Software Foundation» [Fundación para el Software Libre–FSL], una organización de caridad libre de impuestos para el desarrollo del software libre. La FSL también acaparó el negocio de distribución en cinta de Emacs; más adelante lo extendió al agregar otros productos de software libre (tanto GNU como no-GNU) a la cinta, y con la venta de manuales libres.

La FSL acepta donaciones, pero la mayoría de sus ingresos han provenido siempre de las ventas–de copias de software libre, y otros servicios relacionados. En la actualidad vende CD-ROMs de código fuente, CD-ROMs con binarios, manuales agradablemente impresos (todos con libertad para redistribuir y modificar), y las Distribuciones De Lujo (en las cuales incorporamos toda la colección de software lista para usar en la plataforma de su elección).

Los empleados de la Fundación para el Software Libre han escrito y mantenido una cantidad de paquetes de software GNU. Dos notables casos son la biblioteca C y el shell. La biblioteca C de GNU es lo que usa todo programa que corre en un sistema GNU/Linux para comunicarse con Linux.

Fue desarrollada por un miembro del staff de la Fundación para el Software Libre, Roland McGrath. El shell que se usa en la mayoría de los sistemas GNU/Linux es BASH, el Bourne Again SHell(1), que fue desarrollado por Brian Fox, empleado de la FSL.

Hemos provisto los fondos para el desarrollo de esos programas porque el proyecto GNU no se queda solamente en herramientas o un entorno de desarrollo. Nuestra meta era tener un sistema operativo completo, y esos programas eran necesarios para esa meta.

(1) «Bourne again shell» es una broma sobre el nombre «Bourne Shell», que era el shell usual en Unix.

Asistencia para el Software Libre

La filosofía del software libre rechaza una práctica específica de negocio ampliamente difundida, pero no está contra el negocio. Cuando los negocios respetan la libertad de los usuarios, les deseamos éxito.

La venta de copias de Emacs demostró una clase de negocio con software libre. Cuando la FSL se apropió de ese negocio, necesité de otro medio de vida. Lo encontré en la venta de servicios relacionados con el software libre que había desarrollado. Esto incluía la enseñanza, sobre temas tales como cómo programar GNU Emacs, y cómo personalizar GCC, y desarrollo de software, en la mayor parte transportar GCC a otras plataformas.

En la actualidad cada una de esas clases de negocios con software libre está puesta en práctica por una cantidad de corporaciones. Algunas distribuyen colecciones de software libre en CD-ROM; otras venden asistencia en niveles que van desde responder preguntas de usuarios, reparación de errores, hasta el agregado de nuevas características mayores. Incluso estamos viendo compañías de software libre basadas en el lanzamiento de nuevos productos de software libre.

Aunque, tenga cuidado–una cantidad de compañías que se asocian a sí mismas con el término «open source» en realidad basan su negocio en software no-libre que trabaja con software libre. Ellas no son compañías de software libre, sino compañías de software privativo cuyos productos tientan a los usuarios a abandonar su libertad. Ellas usan la denominación «valor agregado» lo que refleja los valores que desearían que adoptemos: conveniencia por encima de libertad. Si valoramos más la libertad, deberíamos denominarlos productos con «libertades sustraídas».

Metas técnicas

La meta principal de GNU era el software libre. Aún en el caso que GNU no tuviese ventajas técnicas sobre Unix, tendría una ventaja social, al permitir cooperar a los usuarios, y una ventaja ética, al respetar la libertad de los usuarios.

Pero era natural que se apliquen los estándares conocidos de buenas prácticas al trabajo–por ejemplo, reservar dinámicamente las estructuras de datos para evitar límites de tamaño fijo arbitrarios, y manejar todos lo posibles códigos de 8 bits cuando tuviese sentido.

Además, rechazamos el enfoque de Unix para pequeños tamaños de memoria, al decidir que no trabajaríamos para máquinas de 16 bits (era claro que las máquinas de 32 bits serían la norma para cuando el sistema GNU estuviese terminado), y al no hacer ningún esfuerzo para reducir el uso de memoria, a menos que excediera el megabyte. En los programas para los cuales no era crucial el manejo de ficheros muy grandes, incentivamos a los programadores a leer el fichero completo en memoria, y luego explorar su contenido, sin tener que preocuparse por la E/S.

Estas decisiones permitieron que muchos programas GNU sobrepasaran a sus contrapartidas Unix en confiabilidad y velocidad.

Computadoras donadas

A medida que la reputación del proyecto GNU crecía, la gente comenzó a ofrecer al proyecto donaciones de máquinas con Unix corriendo. Fueron muy útiles porque la manera más fácil de desarrollar componentes de GNU era hacerlo en un sistema Unix, y luego ir reemplazando los componentes del sistema uno a uno. Pero ellas trajeron una cuestión ética: si era correcto para nosotros siquiera tener una copia de Unix.

Unix era (y es) software privativo, y la filosofía del proyecto GNU dice que no debemos usar software privativo. Pero, aplicando el mismo razonamiento que lleva a la conclusión que la violencia en defensa propia está justificada, concluí que era legítimo usar un paquete privativo cuando ello era crucial para desarrollar un reemplazo libre que ayudaría a otros a dejar de usar el paquete privativo.

Pero, aún cuando esto era un mal justificable, era todavía un mal. En la actualidad ya no tenemos más copias de Unix, porque las hemos reemplazado por sistemas operativos libres. En los casos en que no pudimos reemplazar el sistema operativo de una máquina por uno libre, se procedió al reemplazo de la máquina.

La lista de tareas de GNU

A medida que proseguía el proyecto GNU, se desarrollaron o encontraron una cantidad creciente de componentes, y eventualmente se vio la utilidad de hacer una lista con los huecos faltantes. La usamos para reclutar desarrolladores para escribir las piezas faltantes. Esta lista comenzó a conocerse como la lista de tareas de GNU. Además de los componentes Unix faltantes, agregamos a la lista otros útiles proyectos de software y documentación

que, de acuerdo a nuestra visión, debe tener un sistema verdaderamente completo.

En la actualidad, casi ningún componente Unix queda en la lista de tareas GNU–esos trabajos ya han sido terminados, fuera de algunos no esenciales. Pero la lista está llena de proyectos que algunos pueden denominar «aplicaciones». Cualquier programa que sea atrayente a más de una estrecha franja de usuarios sería una cosa útil para añadir a un sistema operativo.

Aún los juegos están incluídos en la lista de tareas–y han estado desde el principio. Unix incluía juegos, así que GNU debía incluirlos también. Pero la compatibilidad no es un problema para los juegos, así que no seguimos la lista de juegos que Unix tenía. En lugar de ello, listamos un espectro de diferentes clases de juegos que les podrían gustar a los usuarios.

La LPG para Bibliotecas de GNU

La biblioteca C de GNU usa una clase especial de copyleft denominada «GNU Library General Public License» [Licencia Pública General para Bibliotecas de GNU] que da permiso para enlazar software privativo con la biblioteca. ¿Porqué hacer esta excepción?

No es una cuestión de principios; no hay ningún principio que diga que debemos incluir código de los productos de software privativo. (¿Porqué contribuir con un proyecto que se rehusa a compartir con nosotros?) El uso de la LPGB para la biblioteca C, o para cualquier otra biblioteca, es un tema de estrategia.

La biblioteca C hace un trabajo genérico; todo sistema privativo o compilador viene con una biblioteca C. Por lo tanto, el hacer que nuestra biblioteca esté sólo disponible para el software libre, no le daría al software libre ninguna ventaja–sólo hubiera desalentado el uso de nuestra biblioteca.

HAy un sistema que es una excepción a esto: en un sistema GNU (y esto incluye los sistemas GNU/Linux), la biblioteca C de GNU es la única biblioteca C. Así que los términos de distribución de la biblioteca C de GNU determinan si es posible compilar un programa privativo para un sistema GNU. No hay ninguna razón ética para permitir aplicaciones propietarias en un sistema GNU, pero estratégicamente parece que si no se permite, ello hará más para desalentar el uso del sistema GNU que para alentar el desarrollo de aplicaciones libres.

Por estas razones es que el uso de la LPG para Bibliotecas es una buena estrategia para la biblioteca C. Para otras bibliotecas, la decisión estratégica necesita considerarse en cada caso particular. Cuando una biblioteca hace un trabajo especial que puede ayudar a escribir cierta clase de programas, y luego entregarla bajo la LPG, limitándola sólo a programas libres, es una manera de ayudar a otros desarrolladores de software libre, al proporcionarles una ventaja contra el software privativo.

Considere la GNU Readline, una biblioteca desarrollada para proporcionar la edición en la línea de órdenes para BASH. Readline se entrega bajo la LPG GNU ordinaria, no bajo la LPG para Bibliotecas. De esta manera probablemente se reduce la cantidad de uso de Readline, pero eso no significa pérdida para nosotros. Mientras tanto, al menos una útil aplicación se ha transformado en software libre específicamente para poder usar Readline, y ésa es una ganancia real para nuestra comunidad.

Los desarrolladores de software privativo tienen las ventajas que el dinero proporciona; los desarrolladores de software libre necesitan crear ventajas entre sí. Tengo la esperanza de que algún día tendremos una gran colección de bibliotecas cubiertas por LPG que no tengan parangón entre el software privativo, que proporcionen útiles módulos que sirvan como bloques constructivos en nuevo software libre, y que sumen una mayor ventaja para adelantar el desarrollo de software libre.

¿Rascarse una comezón?

Eric Raymond dice que «Todo buen trabajo de software comienza con un desarrollador rascándose una comezón personal». Puede que ocurra algunas veces, pero muchas de las piezas esenciales de software GNU se desarrollaron a los fines de tener un sistema operativo libre completo. Vinieron desde una visión y un plan, no desde el impulso.

Por ejemplo, desarrollamos la biblioteca C de GNU porque un sistema del estilo Unix necesita una biblioteca C, el shell Bourne-Again (bash) porque un sistema del estilo Unix necesita un shell, y el tar GNU porque un sistema del estilo Unix necesita un programa tar. Lo mismo se aplica a mis propios progamas–el compilador GNU C, GNU Emacs, GDB y GNU Make.

Algunos de los programas GNU se desarrollaron para tratar amenazas específicas a nuestra libertad. Por ello, desarrollamos gzip para reemplazar al programa Compress, perdido para nuestra comunidad a causa de las patentes LZW. Proporcionamos fondos para desarrollar LessTif, y más recientemente iniciamos GNOME y Harmony, para lidiar con los problemas causados por cierta biblioteca propietaria (vea más abajo). Estamos desarrollando el GNU Privacy Guard para reemplazar un software popular de cifrado no-libre, porque los usuarios no deben verse obligados a elegir entre privacidad y libertad.

Por supuesto, la gente que escribe estos programas se interesa en el trabajo, y varias personas han agregado muchas características para satisfacer sus propias necesidades e intereses. Pero ése no es el motivo por el cual existe el programa.

Desarrollos inesperados

Al comienzo del proyecto GNU, imaginé que desarrollaríamos el sistema GNU completo, y luego lo entregaríamos completo. No es así como ha sucedido.

Como cada componente de un sistema GNU se implementó en un sistema Unix, cada componente podía correr en sistemas Unix, mucho antes de que existiera un sistema GNU completo. Algunos de esos programas se hicieron populares, y los usuarios comenzaron a extenderlos y transportarlos–a las distintas versiones incompatibles de Unix, y algunas veces a otros sistemas también.

El proceso hizo que dichos programas sean más potentes, y atrayeran tanto fondos como contribuyentes al proyecto GNU. Pero también demoró el completamiento de un sistema mínimo en funciones por varios años, a medida que el tiempo de los desarrolladores GNU se usaba para mantener esos transportes y en agregar características a los componentes existentes, en lugar de adelantar la escritura de los componentes faltantes.

El GNU Hurd

En 1990, el sistema GNU estaba casi completo; el único componente importante faltante era el núcleo. Decidimos implementar nuestro núcleo como una colección de procesos servidores corriendo sobre Mach. Mach es un micronúcleo desarrollado en Carnegie Mellon University y luego en la University of Utah; el GNU HURD es una colección de servidores (o «manada de ñus») que corren sobre Mach, y se ocupan de las tareas del núcleo Unix. El inicio del desarrollo se demoró mientras esperábamos que Mach se entregue como software libre, tal como se había prometido.

Una razón para elegir este diseño había sido evitar lo parecía ser la parte más dura del trabajo: depurar el núcleo sin un depurador a nivel de código fuente para utilizar. Esta parte del trabajo ya había sido hecha en Mach, y esperábamos depurar los servidores HURD como programas de usuario, con GDB. Pero llevó un largo tiempo hacer esto posible, y los servidores multihilo que se envían mensajes unos a otros han sido muy difíciles de depurar. Hacer que HURD trabaje sólidamente se ha tardado varios años.

Alix

El núcleo GNU no se iba a llamar originalmente el HURD. Su nombre original era Alix–denominado así a partir de una mujer que era mi amor de aquella época. Ella era administradora de sistema Unix y había hecho notar que su nombre seguía el patrón de nomenclatura común a las versiones de sistema Unix; a modo de broma, le dijo a sus amigos, «Alguien debería darle mi nombre a un núcleo». Yo no dije nada, pero decidí sorprenderla con un núcleo llamado Alix.

No se dió de esa manera. Michael Bushnell (ahora Thomas), el principal desarrollador del núcleo, prefirió el nombre HURD, y redefinió Alix para referirse a cierta parte del núcleo–la parte que captura las llamadas del sistema y las gestiona por medio del envío de mensajes a los servidores HURD.

Más tarde, Alix y yo nos separamos, y ella cambió su nombre; independientemente, el diseño de HURD se cambió para que la biblioteca C envíe los mensajes directamente a los servidores, y esto hizo que el componente Alix desapareciera del diseño.

Pero antes que estas cosas sucedieran, un amigo de ella encontró el nombre Alix en el código fuente de HURD, y se lo mencionó. Así que el nombre cumplió su objetivo.

Linux y GNU/Linux

El GNU HURD no está listo para el uso en producción. Afortunadamente, está disponible otro núcleo. En 1991, Linus Torvalds desarrolló un núcleo compatible con Unix y lo denominó Linux. Cerca de 1992, al combinar Linux con el sistema no tan completo de GNU, resultó en un sistema operativo libre completo. (La combinación en sí misma dió un considerable trabajo.) Es gracias a Linux que podemos ver funcionar un sistema GNU en la actualidad.

Denominamos a esta versión GNU/Linux, para expresar su composición como combinación de un sistema GNU con Linux como núcleo.

Desafíos en nuestro futuro

Hemos probado nuestra capacidad para desarrollar un amplio espectro de software libre. Esto no significa que somos invencibles o que nada nos puede detener. Muchos desafíos hacen que el futuro del software libre sea incierto; estar a la altura de los mismos requerirá esfuerzos firmes y resistencia, algunas veces durante años. Requerirá la clase de determinación que la gente muestra cuando valora su libertad y no deja que nadie se la quite.

Las siguientes cuatro secciones discuten dichos desafíos.

Hardware secreto

Los fabricantes de hardware tienden cada vez más a mantener las especificaciones de hardware secretas. Esto hace difícil la escritura de controladores libres, y de esa manera, que Linux y XFree86 puedan admitir nuevo hardware. Tenemos sistemas libres completos por hoy, pero no los tendremos mañana si no podemos usar las computadoras del mañana.

Existen dos maneras de lidiar con este problema. Los programadores pueden hacer ingeniería reversa para darse cuenta como usar el hardware. El resto de nosotros puede elegir el hardware que admite software libre;

a medida que nuestro número crezca, el secreto de las especificaciones se transformará en una política contraproducente.

La ingeniería reversa es un trabajo enorme; ¿tendremos los programadores con la suficiente determinación para realizarla? Sí–si hemos construído un fuerte sentimiento de que el software libre es un tema de principio, y de que los controladores no libres son intolerables. ¿Y una gran cantidad de nosotros estará dispuesto a gastar dinero extra, o incluso tiempo extra, para que podamos usar controladores libres? Sí, si se difunde la determinación para tener libertad.

Bibliotecas no libres

Una biblioteca no libre que corre sobre un sistema operativo actúa como una trampa para los desarrolladores de software libre. Las características atractivas de la biblioteca son el cebo; si usted usa la biblioteca, cae en la trampa, porque su programa no puede ser parte útil de un sistema operativo libre. (Estrictamente hablando, podemos incluir su programa, pero no funcionará sin la biblioteca faltante.) Peor aún, si el programa que usa la biblioteca se hace popular, puede hacer caer a otros programadores incautos dentro de la trampa.

La primer instancia de este problema fue el kit de herramientas Motif, allá en los 80s. Aunque aún no había sistemas operativos libres, era claro el problema que Motif iba a causarles más adelante. El proyecto GNU respondió de dos maneras: solicitando a los proyectos individuales de software libre que admitan tanto los widgets del kit libre de herramientas de X como el de Motif, y solicitando a alguien que escriba un reemplazo libre para Motif. El trabajo tomó varios años; LessTif, desarrollado por Hungry Programmers [Programadores hambrientos] tomó la potencia necesaria como para admitir la mayoría de las aplicaciones Motif recién en 1997.

Entre 1996 y 1998, otra biblioteca kit de herramientas GUI no libre, denominada Qt, se usó en una sustancial colección de software libre: el escritorio KDE.

Los sistemas libres GNU/Linux no podían usar KDE, porque no podíamos usar la biblioteca. Sin embargo, algunos distribuidores comerciales de sistemas GNU/Linux que no eran tan estrictos al adherirse al software libre, agregaron KDE a sus sistemas–produciendo un sistema con más capacidades, pero menos libertad. El grupo KDE instaba activamente a más programadores a usar Qt, y millones de nuevos «usuarios de Linux» nunca escucharon la idea de que había un problema con esto. La situación se presentaba lúgubre.

La comunidad del software libre respondió a este problema de dos maneras: GNOME y Harmony.

GNOME, el GNU Network Object Model Environment [Entorno Modelo de Objetos en Red de GNU], es el proyecto de escritorio de GNU. En 1997

Miguel de Icaza lo inició, y se desarrolló con aporte de Red Hat Software, para proporcionar capacidades de escritorio similares, pero usando sólo software libre. Tiene también ventajas técnicas, tales como admitir una variedad de lenguajes, no sólo C++. Pero su propósito principal fue la libertad: evitar el uso de cualquier software no libre.

Harmony es una biblioteca de reemplazo compatible, diseñada para poder hacer funcionar el software KDE sin usar Qt.

En noviembre de 1998, los desarrolladores de Qt anunciaron un cambio de licencia, que cuando se lleve a cabo, hará que Qt sea software libre. No hay manera de estar seguro, pero pienso que esto ocurrió en parte debido a la firme respuesta de la comunidad frente al problema que presentaba Qt cuando no era libre. (La nueva licencia es inconveniente e injusta, así que aún es deseable evitar su uso.)

[Nota después: en Septiembre 2000, Qt fue reeditada bajo la licencia GNU GPL, que esencialmente solucionó este problem.]

¿Cómo responderemos a la siguiente biblioteca no libre que nos tiente? ¿Comprenderá la totalidad de la comunidad la necesidad de mantenerse fuera de la trampa? ¿Alguno de nosotros entregará libertad por conveniencia, y generará un importante problema? Nuestro futuro depende de nuestra filosofía.

Patentes de software

La peor amenaza que enfrentamos proviene de las patentes de software, que pueden colocar a algoritmos y características fuera de los límites del software libre hasta por veinte años. Las patentes del algoritmo de compresión LZW se solicitaron en 1983, y hasta ahora no podemos entregar software libre que produzca GIFs adecuadamente comprimidos. En 1998, se tuvo que quitar de una distribución un programa libre para producir audio comprimido MP3 a causa de la amenaza de un juicio por patente.

Existen maneras de tratar con las patentes: podemos buscar evidencia de que la patente no es válida, y podemos buscar maneras alternativas de realizar el trabajo. Pero cada uno de estos métodos trabaja sólo ciertas veces; cuando ambos fallan, una patente puede forzar a que todo software libre carezca de alguna característica que los usuarios desean. ¿Qué haremos cuando esto suceda?

Aquellos de nosotros que valoremos el software libre por la libertad nos apegaremos al software libre de cualquier manera. Nos las arreglaremos para tener nuestro trabajo realizado sin las características patentadas. Pero aquellos que valoren el software libre porque esperan que sea técnicamente superior, cuando las patentes lo obliguen a mantenerse atrás, es más probable que piensen que se trata de una falla. Por lo tanto, si bien es útil hablar acerca de la efectividad práctica del modelo «catedral» de desarrollo, y de la confiabilidad y potencia de cierto software libre, no debemos detenernos

allí. Debemos hablar acerca de libertad y principio.

Documentación libre

La mayor deficiencia en nuestro sistema operativo libre no está en el software– es la falta de buenos manuales libres que podamos incluir en nuestros sistemas. La documentación es una parte esencial de cualquier paquete de software; cuando un paquete importante de software libre no viene con un buen manual libre, ése es un hueco importante. Tenemos muchos de esos huecos en la actualidad.

La documentación libre, como el software, es un tema de libertad, no de precio. El criterio para un manual libre es muy parecido al del software libre: es una cuestión de otorgar a los usuarios ciertas libertades. La redistribución (incluso la venta comercial) debe estar permitida, en línea y en papel, de tal manera que el manual pueda acompañar a cada copia del programa.

El permiso para modificarlo es también crucial. Como regla general, no creo que sea esencial que las personas tengan permiso para modificar toda clase de artículos y libros. Por ejemplo, no creo que usted o yo estemos obligado a dar permiso para modificar artículos como este, que describe nuestras acciones y nuestra visión.

Pero existe una razón particular debido a la cual la libertad para modificar la documentación es crucial para el software libre. Cuando la gente ejercita su derecho a modificar el software, y agrega o cambia características, si son concientes también cambiarán el manual–así proporcionarán documentación precisa y útil con el programa modificado. Un manual que no permite a los programadores ser concientes y terminar el trabajo, no satisface las necesidades de nuestra comunidad.

La existencia de algunas clases de límites acerca de cómo se deben hacer las modificaciones no implica problemas. Por ejemplo, el requerimiento de preservar el aviso de copyright del autor original, los términos de distribución, o la lista de autores, están bien. Tampoco trae problemas requerir que la versión modificada incluya un aviso de que fue modificada, e incluso que haya secciones completas que no puedan borrarse o cambiarse siempre y cuando dichas secciones traten temas que no sean de índole técnica. Estas clases de restricciones no son un problema porque no impiden al programador conciente que adapte el manual para ajustarlo al programa modificado. En otras palabras, no impiden a la comunidad del software libre la completa utilización del manual.

Sin embargo, debe ser posible modificar todo el contenido *técnico* del manual, y luego distribuir el resultado en todos los medios usuales, a través de todos los canales usuales; si esto no es así, las restricciones obstruyen la comunidad, el manual no es libre, y necesitaremos otro maual.

¿Será que loa desarrolladores de software libre tendrán la conciencia y determinación para producir un espectro completo de manuales? Una vez

más, nuestro futuro depende de nuestra filosofía.

Debemos hablar acerca de la libertad

En la actualidad se estima que hay unos diez millones de usuarios de sistemas GNU/Linux, tales como el Debian GNU/Linux y Red Hat Linux. El software libre ha desarrollado ciertas ventajas prácticas que hacen que los usuarios estén congregándose hacia allí por razones puramente prácticas.

Las buenas consecuencias de esto son evidentes: mayor interés en el desarrollo de software libre, más clientes para empresas de software libre, y mayor capacidad para animar a las compañías a que desarrollen productos de software libre, en lugar de productos de software privativo.

Pero el interés en el software crece más rápido que la conciencia acerca de la filosofía sobre la cual está basado, y esto crea problemas. Nuestra capacidad de enfrentar los desafíos y amenazas que se describieron más arriba depende de la voluntad de mantenerse firmes del lado de la libertad. Para asegurarnos de que nuestra comunidad tiene esta voluntad, necesitamos esparcir la idea entre los nuevos usuarios a medida que ellos llegan a nuestra comunidad.

Pero estamos fracasando en esto: los esfuerzos realizados para atraer nuevos usuarios a nuestra comunidad sobrepasan por lejos a los esfuerzos dedicados a la enseñanza cívica acerca de nuestra comunidad. Necesitamos hacer ambas cosas, y es necesario que mantengamos ambos esfuerzos balanceados.

«Open Source»

La enseñanza acerca de la libertad a los nuevos usuarios se hizo más difícil en 1998, cuando una parte de la comunidad decidió dejar de usar el término «software libre» y usar «open source software» en su lugar.

Algunos de los que favorecieron este término tenían como objetivo evitar la confusión de «free» con «gratis»–una meta válida. Otros, sin embargo, apuntaban a apartar el espíritu de principio que ha motivado el movimiento por el software libre y el proyecto GNU, y resultar así atractivos a los ejecutivos y usuarios comerciales, muchos de los cuales sostienen una ideología que pone las ganancias por encima de la libertad, de la comunidad, y de los principios. Por lo tanto, la retórica de «open source» se centra en el potencial de realización de potente software de alta calidad, pero esquiva las ideas de libertad, comunidad y principio.

Las revistas sobre «Linux» son un claro ejemplo de esto–están llenas de propagandas acerca de software privativo que funciona sobre GNU/Linux. Cuando aparezca la próxima Motif o Qt, ¿incentivarán estas revistas a los programadores a apartarse de ellas, o pondrán propagandas de las mismas?

El apoyo de las empresas puede contribuir a la comunidad de varias maneras; si todo lo demás se mantiene igual, esto es útil. Pero si ganamos su apoyo mediante el recurso de hablar menos de libertad y principio esto puede ser desastroso; hace que empeore el desbalance previo entre el alcance y la educación cívica.

«Software libre» y «open source» describen la misma categoría de software, más o menos, pero dicen diferentes cosas acerca del software, y acerca de los valores. El proyecto GNU continúa utilizando el término «free software» [software libre] para expresar la idea de que la libertad, no solamente la tecnología, es lo importante.

¡Pruébelo!

La filosofía de Yoda («No hay 'para probar'») suena linda, pero no funciona conmigo. He realizado la mayor parte de mi trabajo con ansiedad por saber si podría llevarlo a cabo, y con la inseguridad de que no sería suficiente alcanzar la meta si lo lograba. Pero lo intenté igual, porque no había otro entre el enemigo y mi ciudad. Para mi propia sorpresa, algunas veces he tenido éxito.

Algunas veces he fallado; algunas de mis ciudades han caído. Luego he encontrado otra ciudad amenazada, y me preparé para otra batalla. A lo largo del tiempo, aprendí a buscar las amenazas y ponerme entre ellas y la ciudad, y llamar a otros hackers para que se unan a mí.

En la actualidad, con frecuencia no soy el único. Es un consuelo y un placer cuando veo un regimiento de hackers excavando para mantener la trinchera, y caigo en cuenta que esta ciudad sobrevivirá–por ahora. Pero los peligros son mayores cada año que pasa, y ahora Microsoft tiene a nuestra comunidad como un blanco explícito. No podemos dar por garantizado el futuro en libertad. ¡No lo dé por garantizado! Si usted desea mantener su libertad, debe estar preparado para defenderla.

3.2. IBM aconseja Linux

Linux is known for providing excellent uptime, optimal security and a resistance to attacks that often allow it to run without interruption for as long as the hardware is working.

Benefits of migrating from Windows to Linux:

- Financial
- Security
- Reliability
- Total value

- Future value
- Enterprise-ready
- On Demand

En el servidor de IBM puede leer un artículo explicando los beneficios de migrar: www-03.ibm.com/linux/competitive/windowsToLinux.shtml

Parte II

Desarrollo experimental

Capítulo 4

Estructuras básicas

En una obra grande como tesis o libro conviene ir poniendo etiquetas (label) para luego poder referirse a ellas como ocurre en este capítulo. Ya pusimos una etiqueta en el capítulo 3 y ahora podemos referirnos a él sin miedo a que cambie la numeración: siempre estará todo bien referenciado.

4.1. Listas

Las listas son muy fáciles. No me preguntes si se puede cambiar el estilo de las listas numeradas. Obviamente se puede, pero no es ahora el momento de ocuparse del tema. Busca en internet o en nuestros libros y aprenderás a hacerlo.

1. Uno
2. Dos
3. Tres

Aquí aparecen litas «anidadas» con una construcción transparente. Si sustituyes algunos «enumerate» por «itemize» tendrás otro tipo de listas. Pruébalo. Los editores de LaTeX suelen tener un icono para hacer listas de forma ergonómica, aunque la sintaxis tampoco es tan complicada, ¿verdad? y el resultado es muy bueno.

1. Uno
2. Dos
3. Tres
    a) Tres y cuarto
    b) Tres y medio

c) Cuatro menos cuarto

4. Cuatro

Hay otro tipo más de lista, las listas descriptivas, llamada en lenguaje de LaTeX `description`. Un ejemplo puede verse en la sección 4.5

4.1.1. Citas textuales y similares

Para resaltar una cita textual larga incluida en un documento suelen utilizarse dos procedimientos: enfatizar el tipo de letra, o bien modificar la longitud de línea para el texto citado en relación con la longitud de línea en el texto principal y, en ocasiones, ambas simultáneamente. LaTeX implementa para ese propósito los entornos `quotation` y `quote`.

Ninguno de ellos modifica el tipo de letra. El primero se limita a modificar los márgenes izquierdo y derecho del texto de la cita, mientras que el segundo suprime la sangría de los párrafos incrementando la separación entre éstos.

El texto que sigue, escrito por Bertold Brecht, sirve como ilustración del segundo de dichos entornos.

> Hay personas que luchan un día, y son buenas. Hay otras que luchan un año y son mejores. Hay quienes luchan muchos años, y son muy buenas.
>
> Pero hay algunas que luchan toda la vida: ésas son las imprescindibles.

Si lo copia continuación sustituyendo `quotation` por `quote` podrá observar las diferencias entre ambos entornos.

Aquí se han utilizado estos entornos para incluir una cita textual, pero, obviamente, su utilidad no queda reducida a este caso.

`emph` El comando `\emph` permite *enfatizar textos de un sólo párrafo*, como éste, delimitando entre llaves el texto a resaltar.

4.2. Notas a pie de página y al margen

No hay nada tan sencillo como hacer una nota al pie de página[1] y el resultado queda de lo más profesional. Escribir una nota al margen es también muy sencillo.

Esto es una nota al margen

4.3. Inclusión de gráficos

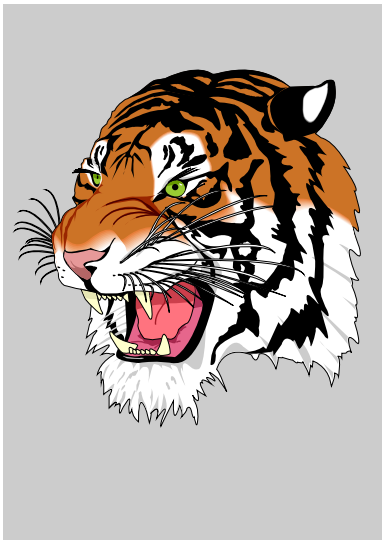
Los gráficos a incorporar han sido creados previamente con alguna herramienta específica, independiente de LaTeX,y haberse guardado utilizando

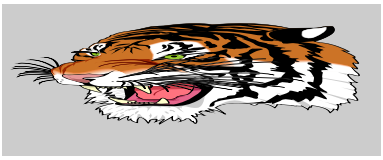
[1] !Lo vés que fácil es!

alguno de los formatos que LaTeX reconoce. ¿Cuales son dichos formatos? Forman una amplia gama, pero los más aconsejables son:

Para LaTeX: el formato postscript encapsulado [.eps] porque a diferencia del formato de mapa de bit [.bmp] mantiene las proporciones al escalar y es suficiente indicar la anchura o la altura. El tipo de documento final que se obtiene puede ser: o bien [.dvi], un formato propio de LaTeX o bien el formato postscript [.ps] que es el utilizado en las imprentas profesionales (hay que compilar con LaTeX en primer lugar y luego utilizar el programa `dvips` sobre el fichero [.dvi] previamente obtenido.

Existen múltiples herramientas que permiten convertir gráficos de un formato a otro. En el CD se incluye alguna de ellas.





Para PDFLaTeX: utilizar formatos [.pdf], [.png] y [.jpg]

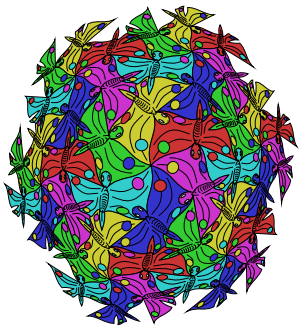
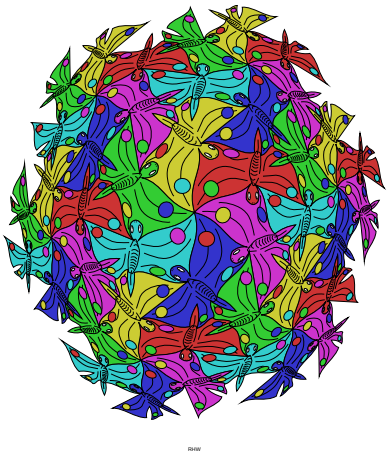


Figura 4.1: Un mosaico de Escher



En los ejemplos anteriores no ha puesto la extensión de los archivos [.eps o .pdf] porque existen archivos con ambos formatos [tiger.eps y tiger.pdf] de ese modo permitimos que LaTeX o PDFLaTeX utilicen el formato que más adecuado le resulte a cada uno de ellos.

4.3.1. Gráficos flotantes con leyenda

Cuando en un documento incorporamos gráficos normalmente querremos que nuestro objeto aparezca junto al texto con el que se relaciona; esto no entrañaría ninguna dificultad si el papel en que se va a imprimir el documento tuviera una «longitud extensible» pero, como este no es el caso puede muy bien ocurrir que el tamaño de la figura haga imposible su ubicación en el espacio que resta para completar la página y entonces sólo cabe colocar la figura en la página siguiente rellenando la página en curso con el texto necesario para completarla. En este sentido necesitaremos que las figuras sean "objetos flotantes". Si se modifica el documento disminuyendo el texto anterior a la figura puede ocurrir que entonces quepa ya en la página, pueden surgir otros problemas cuando se incorpora más texto o cuando aparecen, por ejemplo varias figuras separadas por un texto breve: ¿Cómo distribuirlas? ¿en una misma página o en páginas contiguas, o en una página especial? El entorno `figure` es el adecuado para realizar esa tarea de «flotación» de forma automática permitiendo además incluir leyendas y elaborar una «Índice de figuras» como se ha hecho en este documento

Obsérvese donde ha ido a parar la figura 4.1 que se había incorporado

Figura 4.2: El tigre de Ghostscript

inmediatamente antes de este texto. De no haber utilizado el entorno `figure`, y si el gráfico no le cupiera en el espacio que queda en la página, LaTeX iniciaría una nueva página aunque no hubiera completado la actual.

Hay un parámetro optativo para indicar a LaTeX donde nos gustaría que colocase la figura. En general se hace uso de este parámetro para modificar la solución adoptada por LaTeX, aunque si pedimos imposibles (colocarla en un lugar donde no cabe) LaTeX pasará de nuestros deseos e ignorará el parámetro. En este ejemplo usamos el parámetro optativo `t` (top) para incluir la figura en la parte de arriba de la página; se puede usar `b` (bottom) para incluirla en la parte de abajo, `p` en una página que no contiene texto, únicamente figuras y otros objetos flotantes, o incluso `h` (here) en un sitio preciso,... si puede ser.

Los novatos pueden encontrarse en determinados momentos con la sensación de que LaTeX se rebela y no hace caso a sus indicaciones. Están en lo cierto, porque LaTeX es un profesional y tiene sus propios criterios estéticos aprendidos de impresores profesionales (p.e. nunca coloca dos figuras en la mitad inferior de una página). En la bibliografía sobre LaTeX puede encontrarse más información sobre el tema.

4.4. Tablas

Algunos editores tienen iconos que facilitan la tarea (Kile, TeXMaker, WinEdt) y otros no (TeXnicCenter) pero siempre puede usarse Excel u OpenOfice para construir tablas y luego exportarlas a formato LaTeX. No obstate

Uno	1	one
Once	11	eleven

Cuadro 4.1: Mi primer cuadro con leyenda

conviene saber un mínimo. Observe como construir una tabla básica con tres columnas. Seguro que es capaz de hacer una con cuatro o más jugando además los parámetros l (left) c (center) r (right)

Uno	1	one
Once	11	eleven

La podemos poner centrada y variarla un poco, si queremos

Uno	1	one
Once	11	eleven

Incluso ponerle una leyenda y permitirle «flotar» para mejor adaptarse al espacio disponible en la página como ocurre con el entorno `figure`

Observa bien donde ha ido a parar la tabla 4.1. Le llama cuadro y te molesta. ¡Claro que se puede cambiar! Pero olvida eso ahora: **escribe la tesis**... ya aprenderás a cambiarlas todas con sólo una orden.

Cuando el contenido de uno de los elementos de una columna resulta muy ancho puede ser necesario indicar a LaTeX la anchura de esa columna especial[2] como ocurre en este ejemplo, que además aparece centrado como consecuencia de estar dentro de un entorno `center`

Nombre	**Sexo**	**Éxitos**
Zoftrich	macho	Medalla de oro en 1856 en el open de Sevilla.
Alphex	hembra	Segundo clasificado en Ginebra (1876) y tercero en París (1877).

Una columna de anchura predeterminada se declara con `p{anchura}` como ocurre en el ejemplo que acabamos de ver.

4.5. Índice terminológico

[2]Si se desea, se le puede indicar en todas

El índice general de un libro resulta de suma utilidad para consultar la información que contiene el mismo, pero también es muy útil disponer de un índice terminológico o índice de materias, donde uno pueda buscar una palabra y saber la página donde está el concepto correspondiente y en qué páginas ha sido utilizada.

La elaboración de forma manual de un índice terminológico es una tarea tediosa cuando el documento que está escribiendo es relativamente largo. ¿Y si se hace una repaginación? En este caso no sirve de nada el trabajo realizado y tenemos que rehacer el índice terminológico. LaTeX puede mecanizar esta ingrata tarea mediante con un procedimiento que tiene las siguientes tres etapas:

Seleccionar la información para el índice terminológico Para ello cada vez que se desee que un concepto, un término, una definición, etc. aparezca en el índice haciendo referencia a la página correspondiente usamos el comando `\index{Entrada}` donde `Entrada` es la palabra o frase que deseamos aparezca en el índice. Si esa misma `Entrada` la escribimos en otro lugar del documento y queremos que también vaya al índice, repetimos la misma operación. `index`

Además de `Entrada` se pueden poner, opcionalmente, una `SubEntrada` e incluso una `Sub-SubEntrada` colgando de la entrada principal para ello se utiliza `\index{Entrada!SubEntrada!Sub-SubEntrada}`.

Extraer, agrupar y ordenar esa información Este trabajo lo realiza un programa externo a TeX de nombre `makeindex`. Para ello hay que hacer las siguientes operaciones:

1. En el documento fuente tenemos que:
   - Incluir el paquete `makeidx`; `makeidx`
   - poner el comando `\makeindex` en el preámbulo del documento; `makeindex`
   - poner el comando `\printindex` en el lugar donde queremos que aparezca el índice. `printindex`
2. Compilar el documento fuente.
3. Hacer actuar el programa `makeindex` sobre el archivo `NombreDocumento.idx`.

   Con WinEdt tanto la operación 3 como la 2 se realizan sin más que pulsar en el correspondiente icono[3].

Obtener el índice terminológico deseado Basta para ello volver a compilar.

[3] Observe un ejemplo de referencia cruzada.

A fin de poder ejemplificar la forma de construir un índice terminológico a partir de este momento incluiremos algunos comandos `\index`, obviamente será un índice terminológico muy escaso.

Para conseguir el índice debe proceder como se ha indicado más arriba, en particular ha de incluir en el preámbulo

```
\usepackage{makeidx}
\makeindex
```

y al final, que es donde suelen ponerse los índices de este tipo, el comando `\printindex`. Además, en general, tendrá que hacer el resto de la operatoria, aunque con Kile, TeXnicCenter, WinEdt adecuadamente configurado bastará con que pulse el icono de compilar y estas herramientas se ocuparán por sí mismas de realizar el resto.

4.6. La bibliografía con bibTEX

LaTeX dispone de diferentes soluciones para tratar las referencias bibliográficas. Aquí voy a limitarme a tratar una de ellas: la utilización de bases de datos de bibliografía y del programa `BibTeX`. En mi opinión, si bien es claramente más potente no es sustancialmente más complicada que las otras. El esquema es análogo al de las referencias cruzadas tratado en la sección e igualmente simple:

Base de datos para bibliografías Se dispone de una base de datos dispuesta para la bibliografía. Puede estar construida con anterioridad o se construyendo conforme a las necesidades. El archivo base de datos puede contener los datos de libros, artículos, tesis, trabajos no publicados, etc. identificados como tales, de una forma organizada (que luego se precisará). Frecuentemente será más amplia de lo vayamos a utilizar y en ella podemos incluir todo lo que caiga en nuestras manos y pensemos que "tal vez" nos pueda ser de utilidad. Cada uno de los objetos (libro, articulo,...) tendrá una *etiqueta* (o alias) para identificarlo de forma inequívoca. Puede ser cualquier colección de números y letras no acentuadas.

Referencias a la base de datos Cada vez que en el texto necesitemos referirnos a un objeto de la base de datos para bibliografías lo haremos con el comando `\cite{}` incluyendo entre las llaves la *etiqueta* identificativa del objeto. Si se desea se puede añadir un parámetro optativo entre corchetes para incluir algún comentario adicional (véase para más detalles [2, pág. 316 y ss]).

cite

Preparar la lista de citas Para generar la lista de los objetos citados mediante los diferentes comandos `\cite` se incluye en el lugar que convenga, generalmente al final del texto, lo siguiente

```
\bibliographystyle{plain}
\bibliography{}
```

poniendo entre las llaves del comando `\bibliography` el nombre del archivo donde está la bibliografía (p.e. `BibliografiaTesis.bib`). En ese lugar aparecerán los elementos de la bibliografía *que haya sido citada*. Si se desean añadir objetos que no han sido citados en el texto, se incluye el comando `\nocite{}` incluyendo entre las llaves las correspondientes *etiquetas* de los objetos bibliográficos, separados por comas si son varios.

Generar la bibliografía: compilar+BibTEX+compilar Finalmente se compila el documento, a continuación se ejecuta BiBTEX (en WinEdt puede haber un botón) a fin de que busque en la base de datos de los objetos para bibliografía que han sido citados, los ordene y los incorpore a la Bibliografía concreta de ese trabajo y hecho esto se vuelve a compilar para obtener el resultado deseado[4].

4.6.1. Bases de datos para bibliografía

Sólo resta hablar de la generación de las bases de datos de bibliografía. Comenzaremos diciendo que son archivos "sólo texto" con extensión `.bib` y con una estructura típica de base de datos, es decir, con campos bien delimitados en los que se incluye la etiqueta del objeto en primer lugar, luego el autor, el título, lugar de publicación, fecha, ISBN, etc. Los registros, que así se llaman, no son todos iguales, ya que, por ejemplo, mientras que los libros pueden tener ISBN no ocurre lo mismo con los artículos. De modo que antes de incluir un registro en la base de datos hay que fijar el tipo de objeto (libro, articulo,...) de que se trate.

WinEdt dispone en el menú desplegable "Insertar" de lo que podríamos llamar "fichas modelo" para cada tipo de objeto. Una vez seleccionado el tipo adecuado se van rellenando (o dejando vacíos) los correspondientes campos, comenzando por el primero que es la etiqueta y que debe ser diferente para cada uno de los objetos de la base bibliográfica y ha de rellenarse.

Existen también programas preparados para gestionar de forma amigable estas bases de datos: `BibDb` es uno de ellos que se incluye en esta distribución. No le será difícil aprender a usarlo.

Ahora pondremos aquí [1] otras citas bibliográficas para comprobar el funcionamiento.

[4] Con una configuración adecuada de WinEdt como la que uso yo, simplemente pulsando una vez el botón de compilar se realizan las tres funciones de forma automática.

4.7. Y mucho más

Este capítulo empezó en la página 31 y ahora estamos en la página 40. ¿Ha sido duro el aprendizaje? Con el «editor de textos al uso» que manejas ¿sabes hacer todas estas cosas? ¿Las has hecho alguna vez? ¿Sabes siquiera si puede hacerlas? ¿Cuantas horas sea leyendo manuales o dando zarpazos en teclados y ratones crees que te costaría conseguirlo? ¿Más o menos que leer estas páginas? No espero tu respuesta, este no es un texto interactivo, aunque podría serlo, sólo te pido que dediques un instante a pensar en ellas.

Lo presiento, ergonomía, ergonomía... parpadea en tu mente. No he hecho una revisión concienzuda, pero todo, o casi todo, lo que aquí aparece, está en los iconos o el menú (por si tu memoria se debilita) eligiendo bien el editor de LaTeX. Por contra lo que puede hacer LaTeX no cabe en ningún menu, porque como LaTeX es libre como el aire y no cabe en los barrotes de una jaula. Y es así porque es programable y está abierto al futuro: puede hacer todo aquello que tu puedas pensar. La tesis de M.A. Solano es un ejemplo de ello, pero te pondré dos muy fáciles.

Imaginate que no sabes bien como dar formato a las citas que incorporas de otro lado en tu tesis. ¿Las pongo en itálica? ¿en rojo? ¿con sangría?... Tu tesis tiene 500 páginas ya la has escrito todas en rojo y negrita... Tu director horrorizado (no es para menos) te dice arregláme esto. ¿Cuanto tiempo te llevaría en tu maravilloso editor? El LaTeX, menos de 1 minuto. Juzga por ti. Es sólo un botón de muestra muy simple.

> **Maxima is a system for the manipulation of symbolic and numerical expressions, including differentiation, integration, Taylor series, Laplace transforms, ordinary differential equations, systems of linear equations, polynomials, and sets, lists, vectors, matrices, and tensors. Maxima yields high precision numeric results by using exact fractions, arbitrary precision integers, and arbitrarily precision floating point numbers. Maxima can plot functions and data in two and three dimensions**

Pongo la misma cita pero cambio previamente la definición del objeto «cita»

> *Maxima is a system for the manipulation of symbolic and numerical expressions, including differentiation, integration, Taylor series, Laplace transforms, ordinary differential equations, systems of linear equations, polynomials, and sets, lists, vectors, matrices, and tensors. Maxima yields high precision numeric results by using exact fractions, arbitrary precision integers, and arbitrarily precision floating point numbers. Maxima can plot functions and data in two and three dimensions*

Seguramente tu director te aconsejaría esta última, pero puesta en el preámbulo la definición del nuevo objeto cita sólo tendrías que cambiar una línea para que todas las citas de la tesis cambiaran.

Índice alfabético

Bibliografía

[1] Tim Bienz, Richard Cohn, and James R. Meehan. *Portable Document Format Reference Manual.* Adobe Systems Incorporated, 1996. Disponible en la siguiente dirección de Internet: `http://www.adobe.com`.

[2] Bernardo Cascales, Pascual Lucas, José Manuel Mira, Antonio Pallarés, and Salvador Sánchez-Pedreño. *LaTeX, una imprenta en sus manos.* ADI, 2000.

[3] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion.* Addison-Wesley, 1994.

[4] Joaquín López. Lo que sea. *Israel J. Math.*, 57:23–67, 2008.