El Proyecto GNU

por **Richard Stallman**

publicado originalmente en el libro «*Open Sources*»

La primera comunidad que comparte software

Cuando empecé a trabajar en el Laboratorio de Inteligencia Artificial del MIT (Instituto de Tecnología de Massachusetts) en 1971, formé parte de una comunidad que compartía software que había existido por muchos años. El compartir software no se limitaba a nuestra comunidad en particular; es tan antiguo como las computadoras, del mismo modo que compartir recetas de cocina es tan antiguo como cocinar. Pero nosotros lo hicimos más que la mayoría.

El laboratorio de IA usaba un sistema operativo de tiempo compartido llamado ITS (sistema de tiempo compartido incompatible) que los hackers de la administración del laboratorio (1) habían diseñado y escrito en lenguaje ensamblador para el PDP-10 (Procesador de Datos Programados) de Digital, una de las computadoras más grandes de la época. Como miembro de esta comunidad, un hacker de sistema de la administración del Laboratorio de IA, mi trabajo era mejorar este sistema.

No llamábamos «software libre» a nuestro software porque ese término todavía no existía; pero eso es lo que era. En cualquier ocasión que personas de otra universidad o de una empresa quisieran portar y usar un programa, les dejábamos con gusto. Si veía a alguien usar un programa que no era familiar e interesante, siempre podía pedir ver el código fuente, para que pudiera leerlo, modificarlo, o tomar partes del mismo para hacer un nuevo programa.

(1) El uso de «hacker» para querer decir «intruso en la seguridad» es una confusión provocada por los medios de masas. Nosotros los hackers rechazamos reconocer ese significado, y continuamos usando la palabra para decir «alguien que ama programar y disfruta siendo astuto haciendo eso».

El colapso de la comunidad

La situación cambió drásticamente a comienzos de los años 80 cuando Digital discontinuó la serie PDP-10. Su arquitectura, elegante y poderosa en los 60, no se pudo extender naturalmente a espacios de direcciones mayores que se estaban volviendo factibles en los 80. Esto significó que casi todos los programas que componían ITS se volvieron obsoletos.

La comunidad hacker del laboratorio de IA ya había colapsado, no mucho tiempo antes. En 1981, la compañía derivada Symbolics había contratado a casi todos los hackers del laboratorio de IA, y la comunidad despoblada no era capaz de mantenerse a sí misma. (El libro Hackers, de Steve Levy, describe estos eventos, así como también da una imagen clara de esta comunidad en sus comienzos). Cuando el laboratorio de IA compró un nuevo PDP-10 en 1982, sus administradores decidieron usar el sistema de uso compartido que no era libre de Digital en lugar de ITS.

Las computadoras modernas de esa época, como la VAX o la 68020, tenían sus propios sistemas operativos, pero ninguno de ellos era software libre: debía firmar un acuerdo de no divulgación incluso para obtener una copia ejecutable.

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

La idea de que el sistema social del software privativo, el sistema que dice que no tiene permiso para compartir o cambiar el software, es antisocial, que no es ético, que es sencillamente incorrecto, puede resultar una sorpresa para algunos lectores. ¿Pero qué otra cosa podríamos decir sobre un sistema basado en dividir al público y mantener a los usuarios indefensos?. Los lectores que se puedan sorprender 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 distribuidores de software han trabajado duro durante mucho tiempo para convencer a las personas de que únicamente hay una manera de ver el tema.

Cuando los distribuidores de software hablan de «hacer cumplir» sus «derechos» o de «detener la piratería», lo que en realidad *dicen* 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.

debe aceptarlas de manera acrítica. Así que examinemoslas.

Una de las presunciones es que las compañías de software tienen un derecho natural incuestionable que las habilita para ser dueñas del software, y por lo tanto tener poder sobre todos sus usuarios. (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; los derechos de autor 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 acerca del software es qué trabajos le permite realizar; 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 muchísimo software útil sin ponerle cadenas.

Si nos resistimos a aceptar dichas presunciones, y juzgamos estos temas sobre la base moral ordinaria que nos da el sentido común, poniendo a los usuarios en primer lugar; llegaremos a conclusiones muy distintas. Los usuarios de computadoras deberían ser libres para modificar los programas para ajustarlos a sus necesidades, y libres para compartir el software, porque la base de la sociedad está en ayudar a las demás personas.

No se dispone aquí del espacio necesario para explayarnos en el razonamiento que hay detrás de esta conclusión, por ese motivo pido al lector que vea la página web http://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, firmando acuerdos de no revelación, y prometiendo que no iría en ayuda de mi compañero hacker. Es muy probable que programara software que se entregaría bajo acuerdos de no revelación; incrementando, de esa manera, las presiones sobre otras personas para que traicionen a sus compañeros.

Podría haber hecho dinero de esta manera, y tal vez me hubiese entretenido escribiendo código. Pero sabía que al final de mi carrera miraría hacía los años en los que construí muros para dividir a la gente; y sentiría que usé mi vida para hacer al mundo un lugar peor.

Ya había estado del lado en que se reciben los acuerdos de no revelación, 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 fuese extremadamente frustrante.). Así que no podía decirme a mí mismo que los acuerdos de no revelación eran inocentes. Me enojó mucho cuando él se negó a compartir con nosotros; no podía cambiarme de lugar y hacerle lo mismo a todos los demás.

Otra elección, sencilla 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: ¿existe algún programa o programas que yo pueda escribir, de tal manera de hacer posible una comunidad nuevamente?

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

Como programador de sistemas operativos, tenía las habilidades apropiadas para esta tarea. Así que aún cuando no podía asegurarme el éxito, me di cuenta que había sido elegido para hacer ese trabajo. Decidí hacer al sistema compatible con Unix para que fuera portable, y para que los usuarios de Unix pudieran cambiarse a él con facilidad. El nombre GNU se eligió siguiendo una tradición hacker, como acrónimo recursivo para ,«GNU No es Unix».

Un sistema operativo no implica sólo un núcleo, apenas suficiente para hacer funcionar otros programas. En los 70, todo sistema operativo digno de llamarse así incluía procesadores de comandos, ensambladores, compiladores, intérpretes, depuradores, editores de texto, programas de correo y muchos otros. ITS los tenía, Multics los tenía, VMS los tenía, y Unix los tenía. El sistema operativo GNU los incluiría también.

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

Si yo no me preocupo por mí mismo, ¿Quién lo hará?
Si sólo me preocupo por mí mismo, ¿Qué soy?
Si no lo hago ahora, ¿Cuándo?

Si no lo hago ahora, ¿Cuándo?

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

(1) Como ateo, no sigo a ningún líder religioso, pero algunas veces encuentro que admiro algo que ha dicho uno de ellos.

Libre como en libertad

El término «software libre» a veces se malinterpreta; no tiene nada que ver con el precio [Nota del Traductor: en inglés, «free» en «free software» puede significar «libre» o «gratis», por lo que esta aclaración no aplicaría al español]. Se trata de la libertad. Aquí, por lo tanto, está la definición de software libre: un programa es software libre para usted, un usuario en particular, si:

- Tiene la libertad de ejecutar el programa para cualquier propósito.
- Tiene la libertad de adaptar el programa de acuerdo a sus necesidades (para que esta libertad sea efectiva en la práctica, debe tener acceso al código fuente; porque modificar un programa sin disponer del código fuente es extraordinariamente difícil.).
- Tiene la libertad para redistribuir copias, tanto gratis como por un precio.
- Tiene la libertad para distribuir versiones modificadas del programa, de modo que la comunidad pueda beneficiarse de sus mejoras.

Como «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-ROMs 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, el programa no es software libre.

A causa de la ambigüedad de «*free*» [en inglés], la gente ha estado buscando alternativas, pero nadie ha encontrado una apropiada. El idioma inglés tiene más palabras y matices que ningún otro, pero carece de una palabra simple, que no sea ambigua que signifique «libre», como en libertad. «*unfettered*» (sin cadenas) es la palabra cuyo significado más se aproxima. Otras alternativas como «*liberated*» (liberado), «*freedom*» (libertad) y «*open*» (abierto) tienen un significado incorrecto o alguna otra desventaja.

Software de 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, durante los primeros pasos 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 colección de todo el software de GNU. El sistema GNU incluye programas que no son software de GNU, programas que fueron programados 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 de GNU. Era necesario abandonar el MIT para que el MIT no pudiera interferir con la distribución de GNU como software libre. Si hubiese continuado como parte del personal, el MIT podría haber reclamado la titularidad 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 continuase utilizando las instalaciones del laboratorio.

Los primeros pasos

Poco después de comenzar el proyecto GNU, escuché acerca del *Free University Compiler Kit* (Kit compilador de la universidad libre), también conocido como VUCK (La palabra holandesa equivalente a «libre» comienza con una «V».) Se trataba de un compilador diseñado para manejar múltiples lenguajes, entre ellos C y Pascal, y para admitir múltiples máquinas destino. Le escribí a su autor para consultarle si GNU lo podría usar.

Él me respondió burlescamente, 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 y multiplataforma.

Con la esperanza de evitar tener que escribir todo el compilador por mi cuenta, obtuve el código fuente del compilador Pastel, que era un compilador multiplataforma desarrollado en el Lawrence Livermore Lab. El compilador admitía, y estaba escrito en, una versión extendida de Pascal, diseñada para usarse como lenguaje de programación a nivel de sistema. Le agregué una interfaz para C, y comencé a migrarlo a la computadora Motorola 68000. Pero tuve que abandonar la idea al descubrir que el compilador necesitaba demasiados megabytes de espacio en

la pila, y los sistemas Unix basados en 68000 sólo permitían 64k.

Fue entonces cuando me di 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 «instrucciones» y luego generaba el fichero de salida entero; sin liberar en ningún momento el espacio ocupado. En este punto, llegué a la conclusión de que debería escribir un nuevo compilador 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 la interfaz que había hecho para C. Pero eso pasó unos años más tarde; primero trabajé en Emacs de GNU.

Emacs de GNU

Comencé a trabajar en Emacs de GNU en septiembre de 1984, y a principios de 1985 ya empezaba a ser apto para el uso. 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 tipos de máquinas hasta ese momento.

En esas alturas, la gente comenzó a querer usar Emacsde GNU, lo que priorizó la pregunta acerca cómo distribuirlo. Por supuesto, lo puse en el servidor anónimo de FTP de la computadora del MIT que yo usaba. (A causa de ello, esa computadora, prep.ai.mit.edu, se transformó en el principal sitio de distribución a través de FTP de GNU. Cuando fue retirada 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 debería decirles?

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 una tarifa de 150 dolares. 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 el programa libre para cualquier usuario?

Si un programa es software libre cuando abandona las manos de su autor, esto no significa necesariamente que será software libre para todos los que tengan una copia de él. Por ejemplo, el software de dominio público (software que no está sujeto a derechos de autor) es software libre; pero cualquiera puede hacer una versión modificada privativa de él. Del mismo modo, muchos programas libres están sujetos a derechos de autor pero se distribuyen mediante sencillas licencias permisivas que admiten las versiones modificadas privativas.

El ejemplo paradigmático de este problema es el sistema de ventanas X. Programado en el MIT y publicado como software libre con un licencia permisiva, fue rápidamente adoptado por varias compañías informáticas. Éstas agregaron X a sus sistemas Unix privativos, sólo en formato binario, y lo cubrieron con el mismo acuerdo de no revelación. Estas copias de X no eran tan software libre como lo era Unix.

Los desarrolladores del sistema de ventanas X 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 fueran numerosos.

Esto llevó a una situación paradójica, en la cual dos maneras distintas de contabilizar la cuantía de libertad daban por resultado dos respuestas distintas a la pregunta «¿es libre este programa?». Si usted juzgaba en base a la libertad que se proporcionaba con los términos de distribución de la publicación del MIT, diría que X era software libre. Pero si medía la libertad del usuario promedio de X, tendría que decir que X era software privativo. La mayoría de los usuarios de X usaba las versiones privativas que venían con los sistemas Unix, no la versión libre.

El copyleft y la GPL de GNU

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 de GNU se transformara en software privativo. El método que utilizamos se denomina .«copyleft».(1)

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

La idea central del copyleft es que le damos a cualquiera el permiso para ejecutar el programa, copiar el programa, modificar el programa y redistribuir versiones modificadas; pero no le damos permiso para agregar restricciones por su cuenta. 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 también deben ser libres. Esto asegura que todo trabajo basado en el nuestro quedará disponible para nuestra comunidad si es publicado. Cuando los programadores que tienen trabajo como programadores se ofrecen como voluntarios para mejorar software de GNU, es el copyleft lo que impide que sus empleadores digan: «No puedes compartir esos cambios, porque los queremos usar para hacer nuestra versión privativa del programa».

La exigencia de que los cambios deben ser libres es esencial si queremos asegurar la libertad para cada usuario del programa. Las compañías

La exigencia de que los cambios deben ser libres es esencial si queremos asegurar la libertad para cada usuario del programa. Las compañías que privatizaron el sistema de ventanas X, en general, realizaron algunos cambios para portarlo a sus sistemas y su hardware. Estos cambios fueron pequeños comparados con el gran tamaño de X, pero no fueron triviales. Si 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 que no es libre. Tal combinación inevitablemente no será libre; cualesquiera libertades que falten a la parte que no sea libre, le faltarán también al todo. Permitir tales combinaciones abriría un agujero lo suficientemente grande como para hundir un barco. Por consiguiente, una oblilgación crucial para el copyleft es tapar este hoyo: cualquier cosa agregada a o combinada con un programa bajo copyleft debe ser de tal forma que la versión combinada total sea también libre, y esté bajo copyleft.

La implementación específica de copyleft que usamos para la mayoría del software de GNU es la Licencia Pública General de GNU (GNU General Public License) o GPL de GNU para abreviar. Tenemos otras clases de copyleft que se usan en circunstancias específicas. Los manuales de GNU también están bajo copyleft, pero usan un copyleft mucho más simple, porque no es necesaria la complejidad de la GPL de GNU para los manuales. (2)

(1) En 1984 o 1985, Don Hopkins (un compañero muy imaginativo) me envío una carta por correo. En el sobre había escrito varios dichos entretenidos, entre ellos éste: «Copyleft — todos los derechos reservados». Utilicé la palabra «copyleft» para denominar al concepto de distribución que estaba desarrollando en esa época.

(2) Ahora usamos para la documentación la Licencia de documentación libre de GNU (FDL de GNU).

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 decidimos que era el momento de buscar fondos de nuevo. Por ello, en 1985 creamos la Free Software Foundation (Fundación para el Software Libre), una organización sin ánimo de lucro exenta de impuestos para el desarrollo del software libre. La FSF también acaparó el negocio de distribución en cinta de Emacs; más adelante lo extendió al agregar otros productos de software libre (tanto de GNU como no) a la cinta, y con la venta de manuales libres.

La FSF acepta donaciones, pero la mayoría de sus ingresos siempre han provenido 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, bonitos manuales impresos (todos con la 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 Free Software Foundation han escrito y mantenido una cantidad de paquetes de software de GNU. Dos notables casos son la biblioteca C y la consola. La biblioteca C de GNU es lo que usa todo programa que corre en un sistema GNU/Linux para comunicarse con Linux. Fue programada por un miembro del personal de la Free Software Foundation, Roland McGrath. La consola que se usa en la mayoría de los sistemas GNU/Linux se llama BASHBASH, de Bourne Again SHell (1), que fue programada por el empleado de la FSF Brian Fox.

Patrocinamos el desarrollo de esos programas porque el proyecto GNU no se limitaba solamente a herramientas o a un entorno de desarrollo. Nuestra meta era un sistema operativo completo, y necesitábamos esos programas para esa meta.

(1) «Bourne Again Shell» es un juego con el nombre «Bourne Shell», que era la consola habitual en Unix.

Apoyo 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 demuestra una clase de negocio con software libre. Cuando la FSF comenzó ese negocio, necesité de otro medio de vida. Lo encontré en la venta de servicios relacionados con el software libre que yo había programado. Esto incluía la enseñanza, sobre temas tales como cómo programar Emacs de GNU, cómo personalizar GCC, y programar software, mayoritariamente migrar GCC a nuevas plataformas.

En la actualidad cada una de esas clases de negocios con software libre es practicada por un número de empresas. 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 añadir nuevas características importantes. Incluso estamos viendo compañías de software libre basadas en el lanzamiento de nuevos productos de software libre.

Aunque, tenga cuidado; hay compañías que se asocian a sí mismas con el término «open source» («código abierto»), que en realidad basan su negocio en software que no es 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. Usan la denominación «valor añadido» lo que refleja los valores que desearían que adoptemos: conveniencia por sobre libertad. Si valoramos más la libertad, deberíamos denominarlos productos «libertades sustraídas»

Metas técnicas

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

Pero era natural aplicar 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 soportaríamos máquinas de 16 bits (estaba claro que las máquinas de 32 bits serían la norma para cuando el sistema GNU estuviese terminado), y 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 instalado. Estas fueron muy útiles, porque la manera más fácil de desarrollar componentes de GNU era hacerlo en un sistema Unix, e ir reemplazando los componentes del sistema uno a uno. Pero estas 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 decía 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 ayudara a los demás 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. Si no podíamos reemplazar el sistema operativo de una máquina por uno libre, reemplazábamos la máquina en su lugar.

La lista de tareas de GNU

A medida que proseguía el proyecto GNU, se desarrollaron o encontraron una cantidad creciente de componentes de sistema, y eventualmente se vio la utilidad de hacer una lista con los agujeros que faltaban cubrir. La usamos para reclutar programadores para escribir las piezas que faltaban. Esta lista comenzó a conocerse como la lista de tareas de GNU. Además de los componentes Unix faltantes, agregamos a la lista otros proyectos de software y documentación útiles que, según pensamos, debe tener un sistema verdaderamente completo.

En la actualidad, casi ningún componente Unix queda en la lista de tareas GNU; esas tareas ya han sido finalizadas, con excepción de unos pocos no esenciales. Pero la lista está llena de proyectos que algunos pueden denominar «aplicaciones». Cualquier programa que atraiga a más que unos pocos usuarios sería algo útil para añadir a un sistema operativo.

Incluso hay juegos incluidos en la lista de tareas, y lo han estado desde el principio. Unix incluía juegos, así que naturalmente GNU debía incluirlos también. Pero la compatibilidad no era un problema para los juegos, así que no seguimos la lista de juegos que tenía Unix. En lugar de ello, listamos un espectro de diferentes clases de juegos que les podrían gustar a los usuarios.

La GPL para bibliotecas de GNU

La biblioteca C de GNU usa una clase especial de copyleft denominado GNU Library General Public License (1) (Licencia Pública General para Bibliotecas de GNU) que da permiso para enlazar software privativo con la biblioteca. ¿Por qué hacer esta excepción?

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

La biblioteca C hace un trabajo genérico; todo sistema o compilador privativo 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 privativas 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 LGPL 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, entregarla bajo la GPL, limitándola sólo a programas libres, es una manera de ayudar a otros programadores de software libre, al proporcionarles una ventaja contra el software privativo.

Considere la Readline de GNU, una biblioteca desarrollada para proporcionar la edición en línea de comandos para BASH. Readline se publica bajo la GPL de GNU ordinaria, no bajo la LGPL. De esta manera probablemente se reduce la cantidad de uso de Readline, pero eso no significa una 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 la 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 la GPL que no tengan equivalente entre el software privativo, proporcionando módulos útiles que sirvan como bloques constructivos de software libre nuevo, y que sumen una mayor ventaja para el desarrollo futuro de software libre.

(1) Esta licencia ahora se llama GNU Lesser General Public License (Licencia Pública General Reducida de GNU), para evitar dar la idea que todas las bibliotecas deberían usarla. .

¿Rascarse una picadura?

Eric Raymond dice que «Todo buen trabajo de software comienza con un programador rascándose una molestia personal». Puede que ocurra algunas veces, pero muchas de las piezas esenciales de software de GNU se programaron con el fin 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 Bourne-Again Shell (bash) porque un sistema del estilo Unix necesita una consola, y el tar de GNU porque un sistema del estilo Unix necesita un programa tar. Lo mismo se aplica a mis propios programas; el compilador C de GNU, Emacs de GNU, GDB y Make de GNU.

Algunos de los programas de GNU se programaron para tratar con amenazas específicas para nuestra libertad. Por ello, desarrollamos gzip para reemplazar al programa Compress, perdido para nuestra comunidad a causa de las patentes LZW. Proporcionamos fondos para programar 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 que no es libre, porque los usuarios no deben verse obligados a elegir entre privacidad y libertad.

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

Situaciones inesperadas

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

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 portarlos, a las distintas versiones incompatibles de Unix, y algunas veces a otros sistemas también.

El proceso hizo que dichos programas sean más potentes, y atrajeran tanto fondos como contribuyentes al proyecto GNU. Pero probablemente también demoró la conclusión de un sistema mínimo en funcionamiento por varios años, dado que el tiempo de los desarrolladores de GNU se usaba para mantener nesas migraciones y en agregar características a los componentes existentes, en lugar de continuar, uno tras otro, con la escritura de los componentes que faltaban

El Hurd de GNU

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

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

Alix

El núcleo de GNU originalmente no iba a llamarse HURD. Su nombre original era Alix, denominado a partir de la mujer que era mi amor en aquella época. Ella era administradora de un 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 mantuvo de esa manera. Michael Bushnell (ahora Thomas), el programador principal 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 maneja por medio del envío de mensajes a los servidores HURD.

Al final, Alix y yo nos separamos, y ella cambió su nombre; independientemente, el diseño de HURD se modificó para que la biblioteca C enviara 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 HURD de GNU no está listo para el uso en producción. Afortunadamente, hay disponible otro núcleo. En 1991, Linus Torvalds programó un núcleo compatible con Unix y lo denominó Linux. Cerca de 1992, al combinar Linux con el sistema, que no era tan completo, GNU, resultó en un sistema operativo libre completo (Por supuesto, la combinación en sí misma dio un considerable trabajo.). Es debido a Linux que podemos ejecutar un sistema GNU en la actualidad.

Denominamos a esta versión del sistema GNU/Linux, para expresar su composición como una combinación del sistema GNU con Linux como el núcleo.

Desafíos en nuestro futuro

Hemos demostrado nuestra capacidad para desarrollar un amplio espectro de software libre. Esto no significa que somos invencibles o que nada nos puede detener. Varios 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 secretas las especificaciones de hardware. Esto hace difícil poder escribir controladores libres para que Linux y XFree86 puedan reconocer nuevo hardware. Hoy tenemos sistemas libres completos, pero mañana no los tendremos si no podemos usar las computadoras del mañana.

Existen dos maneras de lidiar con este problema. Los programadores pueden hacer ingeniería inversa para darse cuenta cómo reconocer el hardware. El resto de nosotros puede elegir el hardware que es reconocido por software libre; a medida que nuestro número crezca, el secreto de las especificaciones se transformará en una política contraproducente.

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

(Aclaración de 2008: esta cuestión también se aplica a la BIOS. Hay una BIOS libre, coreboot. El problema es conseguir las especificaciones de las máquinas para que coreboot pueda reconocerlas.)

Bibliotecas que no son libres

Una biblioteca que no es libre que corre sobre sistemas operativos libres actúa como una trampa para los programadores de software libre. Las características atractivas de la biblioteca son el cebo; si usa la biblioteca, cae en la trampa, porque su programa no puede ser parte de un sistema operativo libre (Estrictamente hablando, podemos incluir su programa, pero no **funcionará** sin la biblioteca que falta.). 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 80. 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 los Hungry Programmers (Programadores Hambrientos) tomó la potencia necesaria como para admitir la mayoría de las aplicaciones Motif a principios de 1997.

Entre 1996 y 1998, otra biblioteca de kit de herramientas GUI que no era 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 de 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, es el proyecto de escritorio de GNU. Se inició en 1997 por Miguel de Icaza, y se desarrollo con apoyo de Red Hat Software, GNOME demostró proporcionar capacidades de escritorio similares, pero usando software libre exclusivamente. 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 que no fuese libre.

Harmony es una biblioteca de reemplazo compatible, diseñada para poder ejecutar 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, haría 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 el uso de Qt.)

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

¿Cómo responderemos a la siguiente biblioteca que no sea libre que nos tiente?, ¿comprenderá la comunidad entera la necesidad de mantenerse fuera de la trampa?, ¿o alguno de nosotros entregará libertad por conveniencia, y generará un problema importante?. 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 publicar software libre que produzca GIFs adecuadamente comprimidos. En 1998, se tuvo que quitar de distribución un programa libre para producir audio comprimido en MP3 a causa de amenaza de juicio por patentes.

Existen maneras de lidiar con las patentes: podemos buscar evidencia sobre la invalidez de la patente, y podemos buscar maneras alternativas de realizar un trabajo. Pero cada uno de estos métodos funciona 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 todos modos. Nos las arreglaremos para realizar nuestro trabajo sin las características patentadas. Pero aquellos que valoren el software libre porque esperan que sea técnicamente superior son propensas a llamarlo como un fracaso cuando las patentes retengan su desarrollo. Por ende, si bien es útil hablar acerca de la efectividad práctica del modelo «catedral» de desarrollo (1), y de la confiabilidad y potencia de algunos ejemplos de software libre, no debemos detenernos allí. Debemos hablar acerca de libertad y principios.

(1) Hubiera sido más claro escribir «del modelo de “bazaar”», dado que esa era la alternativa nueva e inicialmente controversia.

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 libre, es un tema de libertad, no de precio [Nota del Traductor: en inglés la palabra usada para expresar «libre» también puede significar «gratis», lo que no es aplicable al español]. 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 también es 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 obligados 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 conscientes también cambiarán el manual. De modo que proporcionen 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 un problema. 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 consciente que adapte el manual para ajustarlo al programa modificado. En otras palabras, no le impiden a la comunidad del software libre la utilización completa del manual.

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

¿Tendrán los desarrolladores de software libre la conciencia y determinación para producir un espectro completo de manuales libres? Una vez más, nuestro futuro depende de la 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 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 por la libertad. Para asegurarnos de que nuestra comunidad tiene esta voluntad, necesitamos esparcir la idea entre los nuevos usuarios a medida que llegan a nuestra comunidad.

Pero estamos fracasando en eso: los esfuerzos 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» - «código abierto»

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 usó «software de código abierto» («open source software» en inglés) 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 dejar de lado el espíritu de principio que había motivado el movimiento por el software libre y el proyecto GNU, y en cambio resultar atractivos a los ejecutivos y usuarios comerciales, muchos de los cuales sostienen una ideología que pone las ganancias por encima de la libertad, la comunidad, y los principios. Por lo tanto, la retórica del «código abierto» se centra en el potencial de realización de software poderoso de alta calidad, pero esquiva las ideas de libertad, comunidad y principios.

Las revistas sobre «Linux» son un claro ejemplo de esto, están llenas de propaganda acerca de software privativo que funciona sobre GNU/Linux. Cuando aparezca la próxima Motif o Qt, ¿incentivarán estas revistas a los programadores a mantenerse alejados de ellas, o pondrán publicidades 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 hablando incluso menos de libertad y principios puede ser desastroso; hace que empeore aún más el desequilibrio previo entre alcance y educación cívica.

«Software libre» y «código abierto» 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 «software libre» para expresar la idea de que la libertad, no solamente la tecnología, es importante.

¡Pruébelo!

La filosofía de Yoda («no hay 'prueba'») suena agradable, pero no funciona conmigo. He realizado la mayor parte de mi trabajo con ansiedad por saber si podría llevarlo a cabo, e inseguro si sería suficiente alcanzar la meta si lo hacía. Pero lo intenté igual, porque no había otro aparte de mí entre el enemigo y mi ciudad. Para mi sorpresa, algunas veces he tenido éxito.

Algunas veces fracasé; 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 mi ciudad, llamando a otros hackers para que vinieran y se unan a mí.

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

volver arriba

Compruebe otras campañas de la Free Software Foundation
Defective by Design, una campaña en contra de la gestión digital de restricciones (o DRM, por sus siglas en inglés).
Windows 7 Sins, el caso en contra de Microsoft y el software privativo.
PlayOgg, apoye a los formatos libres de audio y vídeo.

Última actualización: $Date: 2009/08/02 20:29:00 $

Bosanski [bs]
Català [ca]
Česky [cs]
Deutsch [de]
Ελληνικά [el]
English [en]
Español [es]
Suomi [fi]
Français [fr]
Bahasa Indonesia [id]
Italiano [it]
[ja]
[ko]
Nederlands [nl]
Polski [pl]
Русский [ru]
[zh-cn]
[zh-tw]