Software Libre / Código Abierto

Clase 9 ABR 2018

Hugo J. Curti

(Transcripción/Edición: Elías Todorovich)

Introducción

Vamos a hablar de Software Libre y de Open Source desde un punto de vista conceptual. La parte más técnica las vamos a ver desde un sistema GNU. Vamos a enfocarnos en aspectos históricos. Obviamente Software Libre tiene una parte que es fuertemente ideológica y vamos a hablar de ella. Vamos a hablar también de la implementación práctica. Vamos a ver que Richard Stallman fue el ideólogo que primero habló de esto y después hay muchísima gente que ha trabajado para que esa ideología se pueda implementar y termine siendo una realidad. El proyecto GNU nació a fines de los 1980' o sea que han pasado casi 30 años y el proyecto GNU fue exitoso, se considera que cumplió sus objetivos. "No se puede detener el avance."

Vamos a estudiar dónde empezó eso. A lo largo del curso, vamos a repasar el proceso de booteo, cómo están organizados los directorios en un sistema Unix, lo cual está estandarizado. Como hay un estándar, hay muchas diferencias entre cómo está ordenado Windows y cómo está ordenado en una distribución de Linux. Windows ha ido mejorando mucho pero en otra época era totalmente caótico. De ahí viene que cada programa trae sus librerías, entonces cada programa son 500 MB-1 GB. Eso los chiquitos. En los sistemas libres cuando instalo una aplicación no viene con librerías. Las librerías ya están en el sistema. La ventaja de eso es que cada aplicación son 10-20 MB. Las más grandes pueden tener 200-300 MB porque tienen galerías o imágenes. Las ventajas de eso es que todo ocupa menos lugar, anda más rápido, y está estandarizado. La desventaja es que si eso no anda, ustedes se tienen que encargar de solucionarlo buscando las librerías. Efectos colaterales interesantes: si tengo que armar 3 servidores en Windows armo 3 servidores en 3 máquinas virtuales. En Linux puedo considerar poner las 3 cosas en el mismo sistema porque no hay tanta interferencia. Vamos a ver cómo se configura una máquina para arrancar: init.d, systemd, que son los sistemas que se utilizan para el arranque. Son muy importantes, donde uno controla. Uno enciende la máquina y se disparan un montón de procesos. A veces hay procesos que consumen memoria o tiempo de CPU y no nos interesan; los podemos sacar. La ventaja de los sistemas libres es que uno puede hacer esas cosas de manera documentada. Requiere un esfuerzo que vale la pena asumirlo. Vamos a ver Shell scripting desde el punto de vista de la arquitectura, de dónde viene. Y vamos a terminar viendo tres herramientas muy importantes. Vamos a ver cómo se maneja el compilador de C desde la línea de órdenes. Más allá de manejar el compilador a través de IDEs, hace falta conocer la herramienta cruda. Vamos a estudiar la herramienta make, que es

muy potente y es base para la construcción de proyectos de Software Libre. Finalmente vamos a ver algo de versionamiento: CVS y GIT.

En la práctica vamos a ver cómo funciona el X[*1]. El X es importante, es algo histórico con un contenido muy importante. Por ejemplo X tiene un protocolo vectorizado muy potente que permite levantar una aplicación en una máquina y ver la interfaz en otra mucho antes que NVC o Rdesktop, y además estas otras herramientas pasan mapas de bits. X cierra el lazo de control localmente: el mouse y el teclado están controlados por la máquina local. El dibujo de la pantalla o renderización lo hace el cliente. La aplicación es un cliente de X que se conecta para que el X le muestre la pantalla.

La mirada económica del Software

Cuando aparecieron las computadoras de forma industrial, porque las primeras computadoras eran prácticamente artesanales, ahí empieza la historia. Lo más importante es que no todo el conocimiento de lo que tiene que hacer la máquina está en la misma máquina. Aparece el concepto de Software, que estaba antes de las computadoras pero en las computadoras se vuelve central. La máquina por si misma solo puede seguir órdenes. Entonces el conocimiento que la completa para que haga algo útil es el Software. Cuando las máquinas empezaron a industrializarse tenían que insertarse en la realidad social y económica. En la época anterior, de la Segunda Guerra Mundial, el motivante era ganar la guerra. Cuando esa motivación desapareció, alguien tenía que pagar el costo del avance tecnológico y hasta el día de hoy eso lo paga todo el mundo, sobre todo gente que no lo necesita y lo paga porque está cautiva. Actualizar el teléfono cada tres meses es simplemente un impuesto al avance tecnológico, es la manera que encontraron de financiar el avance tecnológico después de la guerra.

Las primeras computadoras eran muy caras y muy pocas instituciones podían tenerlas, por ejemplo, los mainframe de IBM en los años 1980'. Económicamente hablando estaba por debajo de la producción en masa, y eran muy caras. Al Software no se le prestaba mucha atención económicamente, porque el Software era específico para una máquina. No existía el concepto de portabilidad. Lo valioso era la máquina. Si compras un horno de microondas te viene con el libro de recetas, porque lo que tiene valor es la máquina, no las recetas. Cuando empezó a haber muchas computadoras, nos fuimos dando cuenta que el esfuerzo de escribir un programa merecía crear herramientas para que el programa anduviera en varias máquinas ya sea a nivel de compilación o de interpretación. Entonces apareció el concepto de portabilidad. El lenguaje C fue escrito para eso, para programar con la eficiencia del ensamblado pero no quería casarse con una máquina. El lenguaje C fue creado para incorporar las nociones de programación estructurada y con la idea de escribir un compilador que fuera reorientable, que pudiera generar código hacia distintas arquitecturas. Entonces con un solo programa compilo para Intel, ARM, etc.

El Software empieza a tener relevancia. El costo del Software, que es cada vez más complicado, empieza a tener un peso importante en la ecuación. Ya no es el costo de la computadora. El Software se fue volviendo cada vez más valioso hasta que llegó un momento en el que fue más valioso el Software que la máquina por el esfuerzo en horas/hombre que significaba escribir el Software. Ahí apareció un problema. El sistema capitalista le da valor económico a las cosas y el programa también tenía que tener un valor económico. Puntualmente había que pagarle a los que programaban. Cuando empezó a haber empresas que se dedicaban a hacer Software porque el Software estaba valiendo más que la máquina nació una dificultad muy grande: cómo conectar esto con el sistema económico. El sistema económico maneja bienes escasos y el Software no está regido por las leyes de la materia. El Software no es escaso. El Software tiene un costo alto de creación, pero una vez que está hecho es una idea materializada. ¿Qué pasa con las ideas? Duplicar las ideas tiene un costo prácticamente cero. Eso se volvió un problema para la gente que hacía Software en los años 1970'. Ahí comienza lo que iba a ser un problema de implementación muy grande. En esa época se emitió un manifiesto[*2] que decía que el Software tiene valor, los que programamos Software tenemos que ganar dinero por eso, lo cual está bien. Pero después proponen la siguiente sencilla solución. Vamos a convertir el Software en algo artificialmente escaso. ¿Cómo se hace para convertir el Software en algo artificialmente escaso, porque naturalmente no es escaso? Paso número uno, nos quedamos con el código fuente, no lo entregamos. Al principio eso no era así, cualquiera podía leer el Software. Paso número dos, hacemos entrar al Software a las leyes de propiedad intelectual de la época, lo cual tampoco está tan mal, hasta que dijeron, como nos quedamos con los fuentes y distribuímos los binarios, vamos a hacer una herramienta legal para hacer artificialmente escaso el Software. Ahí es dónde aparecen los números de contrato de licencia. Tomaron el Software como un producto y dijeron el producto se patenta y tiene unas condiciones de uso, no se puede copiar y son un montón de decisiones artificiales que se tomaron para hacer que el Software fuera rentable. Rentable quiere decir que una empresa pudiera ganar mucho dinero con un producto de Software. Lo escribo una vez y lo vendo un millón de veces. Si lo pensamos hoy, eso tiene un aspecto positivo y otro aspecto muy negativo escondido atrás.

Eso empezó a traer problemas enseguida. Mirándolo en retrospectiva, son varios. El principal problema es que es antinatural pero este problema conlleva otros. (1) Por ejemplo hay muchos binarios dando vueltas. ¿Cómo afectó eso a INTEL? INTEL está fabricando sus procesadores actuales que todavía corre código de 16 bits de los procesadores de los años 1980'. Tienen en su diseño cantidad de circuitos que emulan cosas viejas porque todavía hay binarios de los años 1980' que se usan y como no hay fuentes no se pueden compilar. Esos fuentes están perdidos, o las empresas que los generaron están quebradas. Eso pasa mucho con los controladores de PLC que corren en DOS de hace 40 años atrás. Esa es la más inocente de las consecuencias. (2) Otra consecuencia peor es la manera en que se desarrolla el Software. El software privativo no se puede auditar (cómo se que hace lo que dice que hace?). Las empresas aprendieron que el que saca primero un Software, gana. Entonces empezaron a sacar un Software mal desarrollado, mal terminado y eso lo terminó pagando todo el mundo. Hoy la gente está acostumbrada a que se le cuelgue una máquina y pierda datos como si eso

fuera normal. ¿Cómo va a ser normal perder datos? Pasa porque el Software nació así y todavía está fuertemente influenciado por esa tendencia. (3) Además, el software privativo, desde lo económico, concentra, y (4) desde lo social va contra la naturaleza gregaria y la cooperación. En cambio el servicio sí es escaso y es económicamente tratable y sostenible. Además distribuye y no concentra (no genera inflación ni recesión). El Software Libre es de mayor calidad por ser público, más seguro por exposición. La iniciativa de Código Abierto fue un desprendimiento del movimiento del Software Libre y se centró en los beneficios técnicos.

Las cuatro libertades del Software según Stallman

La raíz del problema está en entender el Software como un producto, cuando los productos se rigen por las leyes de la materia y el Software no. El que se dió cuenta de eso fue Richard Stallman. ¿Qué es lo que está mal? Se dió cuenta que estaba enfrentando una hegemonía, una concentración de poder. O sea, alguien estaba tomando un poder sobre él, afectando su libertad. Esa fue la conclusión de Stallman. Ahí se vió que las empresas que producen Software de esta manera restringen la libertad del usuario. Stallman propuso que el Software tenía que tener 4 características muy importantes con respecto a la libertad. Las famosas 4 libertades del Software según Stallman: la primera y fundamental es la libertad de usarlo como yo quiera, no como ellos me dicen. Si yo quiero usar el Software para algo que no fue previsto por los contratos de licencia, por qué no puedo hacerlo, si ni siquiera los perjudico en ninguna forma. Es simplemente una traba legal. La segunda es la libertad de estudiarlo, para lo cual necesito el código fuente. ¿Por qué tengo que poder estudiar el Software? Porque tengo que saber si hace lo que yo creo que hace y no hace otras cosas. Si no sé programar puedo contratar a alguien que lo haga por mí. Puede ser paranoico pensar que el Software trae cosas escondidas pero la potencialidad existe y hay casos documentados. Un caso con un juicio fue el de Sony, que grabó CDs de música con un troyano dentro. Los CDs pueden traer música y un sistema de archivos juntos, puede ser dual. Sony sacaba los discos con una pequeña banda de datos donde venía un troyano. Si lo metías en una máquina con Windows te instalaba un programa que detectaba las firmas en los CDs y no te dejaba reproducirlo si no encontraba la firma para evitar que reproduzcas un CD copiado. Vos metías un CD de Sony y después tu máquina quedaba así. Esas cosas pasan porque está todo escondido. La posibilidad existe que te metan algo en el Software. Hoy está lleno de adware y cosas que parecen inofensivas pero están capturando datos de la gente. Entonces la libertad de estudiar el Software es importante por eso. Si yo quiero tener el control y no que otro lo controle por mí necesito poder estudiarlo. Por lo menos necesito tener el camino abierto. La tercera es la libertad de modificar el Software. Necesito adaptarlo a mis necesidades. Asumir que el producto está terminado y así le va a servir a todos es otro de los grandes errores de estas empresas. Lo que termina pasando es que la gente se adapta porque no le queda otra pero en realidad el Software se tiene que poder adaptar y no al revés, la gente adaptándose al Software, porque una de las características naturales del Software es la flexibilidad. El Software está en el plano de las ideas, no en el plano de la materia. No tiene costo cero hacer la modificación, pero sí tiene costo cero distribuirla y tiene un costo muy bajo hacer una pequeña modificación. A la de usar

el Software, Stallman la llamó libertad cero (se nota que era programador de lenguaje C). Entonces la libertad tres es la de poder distribuir el Software original y las modificaciones. Eso es importante desde lo social. Si a mi algo me hace bien y yo lo puedo compartir para que a otros también les ayude, está genial. No puede ser que a mi me prohiban compartir algo que a mi me sirve y que puede ayudar a otro. Por otro lado, mis modificaciones las hice para mi pero por ahí a otro también le sirven. ¿Por qué no puedo distribuirlas? Stallman dice que esas cuatro condiciones son las que hacen que el Software sea libre. Eso se materializa en algún contrato de licencia que lo especifique literalmente, aclarando siempre quién es el autor. El tema de la propiedad intelectual no se discutió nunca. Si yo escribí algo está bien que le ponga que yo lo escribí. El problema es la interfaz del Software como producto, el pensar el Software como producto.

Stallman no se quedó con esto solo, el los 1980' armó una idea de cómo se debía hacer negocio con el Software diferente de la que estaba de moda en la época. El Software Libre no era solamente una cuestión ideal y nada más. Esa manera de hacer negocio estaba mucho mejor que la que estaba instalada en la época. Stallman propuso que tratar y vender al Software como servicio, no como producto. Nadie habla de que al Software hay que regalarlo. Nadie tiene que trabajar gratis, no es la idea. La idea es cambiar el modelo de negocio. Entonces, yo escribo un Software para prestar un servicio. Lo que vale es el servicio, no el Software. El servicio es el valor agregado que yo le pongo. Alguien necesita una Base de Datos, yo le instalo una base de datos, la adapto a sus necesidades, y me voy a quedar con él hasta que le ande, que la entienda y hasta que esté satisfecho. Y le voy a cobrar por el servicio. Mañana otro necesita una Base de Datos parecida a la anterior, pero no es la misma. Voy a hacer el trabajo de nuevo y voy a cobrar otra vez. Lo que propone Stallman, es poner la Base de Datos en el repositorio para que cualquiera preste un servicio similar. "Las ideas grandes, que han llegado muy lejos, no llegaron lejos porque alguno se las guardó." El Software Libre nunca es un producto terminado. El Software privativo es un producto terminado. Al Software Libre, alguien tiene que darle el valor agregado final. Por eso el Software Libre en su contrato de licencia dice que no trae garantía de mercantibilidad, de rentabilidad, ni de adaptación a uso específico. Uno de los términos muy importantes del Software Libre es ese. Esa garantía o te la provees vos mismo o te buscas alguien que te la provea. Vos tenés que hacer la última parte del trabajo siempre. En un producto terminado pago, y teóricamente me olvido. Lo cual es mentira, termino adaptándome al producto. Lo que propone Stallman es algo mucho más amistoso, más personalizado. En el conjunto es mucho mejor, implica mucho menos esfuerzo individual. Es una sinergia muy grande y todo se hace más barato así.

Ahora vamos a poner un ejemplo del modelo privativo y del modelo de Stallman. Imaginemos que quisiera desarrollar un cliente de correo electrónico según el modelo privativo. Arranco de cero, necesito programar desde el cliente de SMTP, el cliente de IMAP, toda la librería gráfica, toda la interfaz, toda la funcionalidad. Imaginemos que estamos en los orígenes, hoy se consigue todo hecho. Entonces tenía que conseguir mucho dinero para contratar programadores para que lo hagan. Si había alguna librería, era privativa y la tenía que pagar. Tenía que buscar un inversor, y asumir el riesgo. Tenía que escribir el programa y esperar que funcionara. Quizás no anda. El riesgo es enorme, para el inversor también, y por eso va a

poner condiciones. Una vez que el Software está terminado tengo que recuperar la inversión. Obvio que lo voy a empaquetar y le voy a poner un número de licencia y voy a lanzar una campaña publicitaria para que me lo compren. Así está vendido como producto. Si tengo suerte y se vende mucho, recupero la inversión y gano algo. Después espero más ganancias. A lo sumo tendré que hacer actualizaciones, mantenimiento. Al final la idea es hagámoslo una vez y vendámoslo mil veces. Es la idea de “la pego con algo y me salvo” y no tengo que trabajar más. Eso se ve desde la época de la búsqueda de oro hasta las punto com en la década del 2000'. Lo cierto es que eso es lo mismo que jugar a la lotería. No puede toda la humanidad vivir así. Ese modelo no es sostenible y genera escasez. Unos se suben a los hombros de otros. Este otro modelo que propone Stallman sí es sostenible. Voy a desarrollar mi cliente de correo electrónico basado en Software Libre. No necesito inversor ni dinero, ni empleados. Me busco los componentes libres que necesito y los uno. Los clientes SMTP, IMAP, la librería gráfica, todo está en el repositorio. Una vez que lo tengo, se lo entrego al que me lo pidió y me paga por el servicio de habérselo armado. La ganancia es menos pero la inversión también es mucho menos. Muchas veces nos meten en la cabeza que el Software es el producto, lo que tiene el valor, y el Software en definitiva es una herramienta para brindar un servicio. No es el producto final.

El Software a medida puede o no ser libre. Si te contratan para hacer un Software, el que te contrató probablemente se rija con el paradigma privativo y no hay muchas opciones. No podés pedirle a alguien que hizo una inversión grande, que de repente reparta el código fuente. El copyright puede ser compartido entre el autor y el que te contrató.

En el paradigma privativo te vas a enojar si te copian el Software, o vas a mirar para otro lado y que la gente quede en la clandestinidad. La persona que está clandestina es más fácil de convencer de cualquier cosa. Cuando entras en la clandestinidad tu posición de negociación ante cualquier cosa es más baja. Me instalé el programa copiado y no anda. Y bueno, no anda pero es copiado, ¿qué se le va a hacer?. Si lo instalas legalmente, llamas, protestas. Entonces, vas degradando tu calidad de vida hasta niveles terribles.

La Fundación de Software Libre (FSF)

Richard Stallman creó la Fundación de Software Libre, que fue el soporte legal y técnico del proyecto GNU. El proyecto GNU es, básicamente, este repositorio del que les estaba hablando. Es la idea de tener un gran repositorio de Software para hacer todo tipo de cosas protegido legalmente por una licencia de Software libre. GNU quiere decir GNU is not UNIX. Es un acrónimo recursivo, un chiste de programador. Aparte GNU en inglés se pronuncia como new y como el animal que se usa en el logo del proyecto. Cuando comenzó GNU, como adoptó la filosofía de UNIX, todo el mundo decía que era UNIX. Stallman se cansó de decir eso no es UNIX y por eso le puso GNU is not UNIX. UNIX no fue un Software Libre, sino uno de los más privativos que hay por las patentes que tiene.

GNU nació con unas cuantas herramientas básicas, muy importantes que el mismo Stallman empezó a desarrollar. Después consiguió fondos, a través de la Fundación, para pagarle a programadores. Lo primero que armó fue un editor de texto muy potente que es el EMACS,

armó un compilador de C, el GNU Compiler Collection o GCC, armó la librería GNU C Library (conocida como glibc) y comenzó a programar el kernel de un sistema operativo que se llama GNU Hurd, que todavía existe. Lo que pasa es que Stallman adoptó la arquitectura microkernel para el SO demasiado temprano en su época. El punto débil de microkernel es que es lento. Va a la velocidad que puede ir su mecanismo de IPC (inter-process communication). Stallman tuvo muchísima dificultad con el SO. Todo lo demás iba bien.

Stallman también hizo la parte legal para proteger al Software Libre. Sacó la GNU Public Licence (GPL) que protegía a todo el Software del proyecto GNU. Es una licencia que tiene unos cuantos términos legales y provee las cuatro libertades del Software. No hace falta que distribuyas los fuentes, pero se exige tener acceso al código fuente con el que construiste el binario que te están dando. También dice que no se puede limitar el uso del Software. Después tiene una cláusula muy discutida que se llama de copyleft (otro chiste de programador, left es izquierda y también es el pasado de dejar). La cláusula de copyleft dice que todo programa devenido de un programa GNU debe tener una licencia igual. Es muy fuerte, muy discutido y causó muchos problemas. Dice que es ilegal agarrar un programa GNU, cerrarlo y venderlo.

La licencia GPL tenía un problema con las librerías. Había gente que quería hacer Software privativo sin basarse en Software GNU pero como quería correr en el sistema GNU tenía que vincularse con las librerías del sistema GNU. Eso era ilegal también. Eso se relajó después y se lanzó una licencia Library GPL (GNU Lesser General Public License, LGPL) que permite vincular con librerías GNU aunque tu Software sea privativo. Fue una medida muy importante para hacer que las cosas bajen a la realidad. La ideología es hermosa pero si no se relajan cosas en el momento de implementar, se queda en el plano ideal. Para que las cosas se puedan bajar a la realidad, en el proceso de implementación hay que tener en cuenta el estado de realidad del mundo y adaptarse. Si no, no baja. Cuando empezaron con el Software Libre, y les fueron a hablar a las empresas sobre el Software Libre, y les hablaban de los usuarios y la libertad, los gerentes formados en la vieja escuela, decían ¿esto que ganancia me da? Entonces no se estaba adaptando al estado de la época. Los ideales no deben negociarse pero las implementaciones se deben negociar porque si no el ideal no se realiza. Negociar el ideal es peligroso porque perdés el rumbo. No todos podemos ser Stallman, si todos fuéramos Stallman, el Software Libre no hubiera llegado a ninguna parte. Los que no somos Stallman tenemos que negociar con la realidad para lograr algún objetivo sin quedarnos paralizados.

Código abierto (Open Source)

Un grupo de gente que al principio estaban con Stallman en el Software Libre empezaron a decir: esto de las libertades y la filosofía no vende nada, nos mandan al diablo, ¿qué hacemos? Entonces ellos se concentraron en los aspectos técnicos: el Software madura más rápido, todos lo pueden revisar. Vamos a mostrar los beneficios que tiene trabajar con el Software abierto. Así nació la iniciativa de Código Abierto (Open Source). Código Abierto no es lo mismo que Software Libre pero en general las licencias de Software libre y Código Abierto son

compatibles entre sí. Código Abierto se enfoca más en lo pragmático, en las ventajas técnicas, y deja un poco de lado los usuarios y sus libertades. Con eso lograron penetrar más.

La comunidad de Código Abierto tienen mecanismos de protección. Cuando ve peligro de cerrar un proyecto abre un fork. Hay un capítulo de Los Simpsons que representa la idea con una alegoría muy evidente. Homero toca una tecla y se arma un sitio con un millón de visitas. Vino uno muy parecido a Bill Gates a comprarle el sitio, le dio un cheque y le dijo Homero -yo no me vendo. Entonces le puso tres ceros más y Homero le dijo -así sí. El comprador dijo -compren muchachos! Entraron tres tipos con palos y le rompieron todas las computadoras. Se trata de comprar proyectos para destruirlos, y la forma de defenderse es mediante forks.

Con grandes distribuciones de Linux pasó. Redhat, por ejemplo, en un momento comenzó a coquetear con el Software privativo y automáticamente salió Fedora. SUSE empezó a coquetear con el Software privativo y salió Open SUSE. Debian se mantuvo siempre indemne. Debian tiene un contrato social desde que nació. Se manifestaron a favor del Software Libre y el Código Abierto. Debian es la distribución más grande y la madre de muchas otras como Ubuntu.

Las licencias de Código Abierto comparten con las de Software Libre en que exigen el código fuente. Algunas tienen cláusula de copyleft, otras no. La licencia BSD actual no tiene copyleft. Eso quiere decir que nadie te prohíbe que cierres el Software y lo vendas como producto. Algunos alegan que la libertad es libertad en todos sus términos, con lo cual si te digo que podés usar el Software para lo que quieras pero no lo cierres y lo vendas, estás yendo contra la libertad. Lo cierto es que la cláusula de copyleft era necesaria en el arranque para proteger algo que estaba naciendo para que no lo pisaran. Hubo varios ataques terribles contra el Software Libre. Hoy la mayoría de las empresas no ven más al Software Libre como una amenaza. Más bien hoy tratan de aprovecharlo en vez de atacarlo. Google no es una empresa de Software Libre pero adoptó mucho Software Libre y muchas de las cosas que hacen las liberan. Es una empresa que tiene un paradigma que para ser corporación está bastante más cercano al Software Libre que otras empresas más duras como Microsoft.

Una licencia típica comercial dice que el Software es para usarse en una sola máquina, que está terminantemente prohibido copiarlo excepto con fines de backup, está terminantemente prohibido desensamblar el Software, modificar el Software, prohibido usar el Software para cualquier otra cosa que no sea para lo que está pensado.

GNU Linux

Linus Torvalds es el autor del kernel de Linux original. Torvalds es un finlandés ex-alumno de Andrew Tanenbaum. Tanenbaum enseñaba SO y tenía una implementación de UNIX libre que se llama MINIX. MINIX era una pequeña implementación para uso didáctico que escribieron en

el grupo de Tanenbaum. Tanenbaum quería armar un planificador de CPU aprovechando todas las características que tenía la Intel 80386 (conocida como 386). Era la primer CPU de intel que salía con binario de 32 bits y soporte hardware para multitarea real. Torvalds armó un planificador de CPU y lo publicó en Usenet[*3].

La primer versión estable de Linux, Kernel 1, salió en 1992. Después de que se desprendieron de MINIX, y después de que aportaron un sistema de archivos: una implementación del sistema de i-nodos que es el ext2. Eso más drivers de tarjetas de red, de terminales, etc. El primer Linux tenía los códigos fuente y había salido con una licencia rara, para uso académico. Cuando vio que su kernel iba muy despacio, Stallman fue a hablar con Torvalds. Ahí Stallman convence a Torvals de que le cambie la licencia por una GNU y el sistema GNU adoptó el kernel de Linux. Ahí nació lo que se llama GNU/Linux. Linux es el kernel, GNU es una familia de programas. Después compilaron Linux con el compilador GNU. Ahí se armó el triángulo de base: sistema operativo, librería de C y compilador. Si se reorienta este triángulo puedo pasar a otra arquitectura. Si aparece una arquitectura nueva, ARM por ejemplo, tenemos que reorientar el compilador para que genere el binario de ARM. Tenemos que reorientar el kernel, para que utilice las herramientas que provee ARM para la administración. Y tenemos que reorientar la librería de C para que ajuste sus constantes a los anchos de palabra y todo de ARM. Una vez que tengo esas tres cosas, arriba van todos los programas del sistema GNU y ya tengo un sistema GNU en una arquitectura nueva. Hicimos esa experiencia hace unos años con la SUN. Fue una de las primeras experiencias en el mundo porque salimos casi juntos con otro grupo que hizo lo mismo. Hay que tener mucha experiencia para hacer este trabajo. Por eso aparecieron las distribuciones.

Distribuciones

Una distribución es una organización que arma la distribución. Armar la distribución es juntar todos los fuentes, compilarlos, armar un sistema de archivos completo con todo el sistema, cerrarlo y ofrecerlo mediante ftp, http, CDs, etc. Las distribuciones son parecidas porque el Software que está debajo es el mismo. La distribución aporta el trabajo de armarlo, y también tiene alguna orientación ideológica o técnica. Por ejemplo, Slackware es una de las más antiguas. Es una distribución sencilla que simplemente armaba los binarios y al principio no tenía sistema de administración de paquetes ni sistema de actualización. Debian nació del movimiento de Código Abierto y se comprometió desde el principio a mantener los ideales tanto del Código Abierto como del Software Libre. La palabra Debian viene de la contracción de los nombres Debora y Ian (Ian Murdock). Ellos armaron la distribución Debian con el propósito de mantener ese contrato social. Es una distribución grande. Tiene unas políticas muy estrictas de trabajo. Trabajar en Debian es prácticamente un proceso iniciático. Cualquiera puede contribuír pero alguien de adentro te tiene que tomar la contribución. Debian creó un sistema de administración de paquetes muy bueno y fue la primera que incursionó en el tema de mantener el equilibrio de la madurez de Software.

Después aparecieron SUSE y Redhat, SUSE en Europa y Redhat en EEUU; las dos con el mismo objetivo, noble al principio pero peligroso después: hacer fáciles las cosas para que el usuario final tenga acceso. Debian no tenía ese objetivo y Slackware tampoco. SUSE y Redhat se preocuparon por armar instaladores, configuraciones más acabadas, como para un usuario prácticamente instale el sistema y salga andando. Hoy ya viene todo hecho pero en Software Libre hay que ensuciarse un poquito las manos a veces y meterse. Eso es importante porque nos permite poner la consciencia en lo que hacemos, cosa que otro Software no nos permite. Sabemos lo que hacemos, no estamos trabajando a ciegas. Llegamos a aprender cómo son las cosas desde cero, que es otra consecuencia de la libertad. La libertad implica consciencia, implica esfuerzo. No es gratis la libertad. En Software Libre hay que salirse del confort. Yo fui usuario de Windows 95. Cuando empecé a trabajar con Linux me di cuenta de que Linux podía hacer lo mismo pero tenía la documentación, podía entender lo que hacía, podía ir más lejos. En Windows siempre era prueba y error, tratar de robar información de algún lado, porque no había documentación. En Linux podías investigar, meterte en un universo con miles de cosas, y aprender un montón.

UNIX

UNIX fue el sucesor de Multics. Multics era un proyecto muy grande de SO y UNIX es una versión más simple.

UNIX, originalmente, estaba pensado para un solo usuario. Después se le agregó soporte de multi-usuario. Multics es un dinosaurio pero hizo un aporte muy importante: es un SO armado como arquitectura de filtros y tuberías. Unix es un kernel monolítico bien antiguo y la línea de comandos adoptó la arquitectura de filtros y tuberías. Esto es el concepto de tener muchos programas de uso específico y un sistema para conectarlos con facilidad. UNIX es un Software privativo que nació en los Bell Labs y fue cambiando de manos. Minix fue el proyecto de Tanenbaum copiando a UNIX desde cero para enseñar. Linux fue el trabajo de Torvalds, que empezó colgado de MINIX y después se desenganchó. También es un clon de UNIX. Seguía los mismos patrones, las mismas lógicas. El objetivo era que se pudiera compilar un programa para UNIX y funcionara. Entonces fueron adoptando todas las mismas interfaces, las mismas llamadas al sistema, los mismos comandos. Un grupo en la Universidad de Berkeley sacó una versión de ellos que también es libre, el BSD (Berkeley Software Distribution). Hay varias versiones: el FreeBSD, OpenBSD, NetBSD, DragonFly BSD, Darwin, y TrueOS. Pero tiene una diferencia ideológica importante con el Linux. BSD está desarrollado por un grupo cerrado. Linux acepta aportes, que pasa por muchos controles, pero los puede hacer cualquiera. Linux llegó mucho más lejos pero tuvo sus reveses en la historia. Linus Torvalds fue siempre muy cuidadoso en lo que se refiere al ordenamiento del mapa de memoria del sistema, a las políticas para aceptar distintos controladores para aceptar en el fuente y con todo lo que tiene que ver con las rutinas de testeo de regresión. Fue fundamental en una arquitectura monolítica, porque si no arreglabas una cosa y rompías otras diez. Para evitar eso tiene una rutina muy cuidadosa de testeo de regresión, que se ejecuta cada vez antes de lanzar una nueva versión.

Ejecuta testeos de todo el sistema desde que empezó hasta ahora, está horas ejecutando. Es una manera que tiene de asegurar que no salgan errores feos. Se han reportado esporádicamente fallas de seguridad, algún bug medio feo. Muchísimos menos que los que puede tener Windows aunque nadie se entera.

POSIX

POSIX es un estándar. Después que había varios UNIX dando vueltas, llegó el momento de unificar, de hacer un estándar para que todos se acoplen a ese estándar y los programas puedan ser portables. POSIX estandariza los comandos más importantes, las llamadas al sistema de los sistemas UNIX. Todos los sistemas tienen una interfaz POSIX hoy en día. Una excepción muy importante es Windows, pero Windows ha hecho un portado de las librerías de GNU a Windows. El más conocido es el Sygwin que ofrece una interfaz POSIX al Windows. Entonces, podés ejecutar una shell en Windows y tiene mecanismos para conectar y poder interactuar a nivel de POSIX en Windows. POSIX modela threads, procesos, llamadas al sistema, comandos, opciones de comandos. Hay muchas cosas de UNIX clásico reguladas por POSIX.

Del POSIX vienen las primeras librerías de C que traían un problema muy serio con la locación de memoria. Por ejemplo, la función gets en C (get string). La función get string clásica tomaba un puntero a un buffer para alocar un string que podía entrar por std input y la función iba leyendo hasta que alguien apriete un enter y ahí cortaba. El problema es que yo no podía saber de antemano el tamaño del buffer que yo le tenía que pasar. Era una función muy peligrosa porque yo hacía un buffer de 40 y si alguien escribía más de 40 caracteres la función seguía y hacía un overflow de memoria. Había muchas funciones como esa y que hasta hace poco estaban por todos lados. En nuevas versiones de POSIX, esas funciones se fueron reemplazando por funciones seguras. Funciones que tomaban, por ejemplo, como parámetro el número como cota para el buffer, si te pasas, te corto.

Linux es un clon libre del viejo UNIX, como también lo es BSD, como también lo fue en su momento el SunOS, el Solaris, Irix, etcétera; algunos eran libres y otros privativos. Todos eran UNIX-like y todos adherían al POSIX de manera que los fuentes de uno, si los compilabas en otro, andaban. Esta noción de portabilidad a nivel de código fuente estaba bastante bien lograda. No es la portabilidad que buscaba Java, era a nivel de binario directamente, que esa es otra cosa.

Referencias

[*1] Wikipedia: X Window System (X11, o simplemente X) es un sistema de ventanas para sistemas operativos Unix-like.

[*2] Wikipedia: Open Letter to Hobbyists, 1976, https://en.wikipedia.org/wiki/Open_Letter_to_Hobbyists

[*3] Publicado en el newsgroup "comp.os.minix”