UNIX: HISTORIA, FILOSOFÍA Y ESTÁNDARES.

HISTORIA

Partimos de la década de los 60. Los Beatles triunfan con *Please, please me;* el presidente Kennedy es asesinado en Elm Street; los jóvenes experimentan con el LSD en el festival de Woodstock; Elvis Presley es el rey del Rock and Roll y España experimenta un crecimiento económico sin precedentes durante la época franquista. Eran los 60 y los laboratorios AT&T Bell junto con el MIT trabajaban en un sistema operativo experimental llamado Multics (**Mult**iplexed **I**nformation and **C**omputing **S**ervice), diseñado para funcionar en un GE-645, un potente ordenador de aquella época. Las pruebas de rendimiento de las primeras versiones de este sistema no fueron tan bien como se esperaba por lo que Bell Labs paralizó el desarrollo. Ken Thompson y Dennis Ritchie, que eran los principales responsables del proyecto, se quedaron sin nada que hacer y victimas del aburrimiento decidieron crear un juego, el Space Travel**,** que funcionaba sobre aquel GE-645. El jueguecito les gustaba, pero tenia el inconveniente de que solo se podía jugar en aquel "supercomputador" tan caro así que decidieron portarlo a una máquina más pequeña, una PDP-7. Eso no era tarea fácil ya que por aquel entonces todo se desarrollaba en lenguaje ensamblador y para conseguir su objetivo había que hacer cambios en la gestión de memoria, añadir soporte para números reales, ajustarse al sistema de ficheros, etc. Tanto se enzarzaron en esta tarea que acabaron creando un nuevo sistema operativo, el UNICS. AT&T Bell, que vio que la cosa podía tener futuro decidió subvencionar el desarrollo incorporando a varios programadores, entre ellos a Brian Kernighan. UNICS pasa a llamarse UNIX y va adquiriendo nuevas funcionalidades y pequeños programas que se incorporan al sistema como comandos. En 1971 ya se podían usar, por ejemplo, cat, chmod, chown, cp, ls, mv, wc, who, roff...

El desarrollo era rápido pero había algo que lo frenaba. Cansados de programar en ensamblador, Kernighan y Ritchie decidieron crear un nuevo lenguaje de programación de más alto nivel, añadiendo nuevas funcionalidades al ya existente lenguaje B. El resultado fue C, un lenguaje que permitía abstraerse de las particularidades de la arquitectura subyacente pero al mismo tiempo trabajar a nivel de bit y gestionar los recursos de memoria. Su potencia y elegancia impresionaron a todo el equipo de desarrollo por lo que en 1973 se decide reescribir en C todo UNIX. Muy poco después, en 1974, Ritchie y Thompson presentaban a la comunidad su sistema a través de un artículo publicado en la prestigiosa revista *Communications of the ACM,* donde hablaban sobre el diseño de Unix y sobre las 600 máquinas que por el momento funcionaban con él. Es entonces cuando el sistema operativo comienza a ganar popularidad y su desarrollo se acelera vertiginosamente. Surgen múltiples versiones y se porta a una infinidad de arquitecturas distintas. Las universidades empiezan a utilizarlo como plataforma de

trabajo para la investigación, grandes empresas como bancos y aseguradoras lo emplean como base de sus sistemas y hasta los ejércitos comienzan a experimentar con él para adaptarlo a sus necesidades. AT&T introduce su Unix System 3, la universidad de Berkeley desarrolla el BSD v4 y algunos vendedores como Interactive Systems, Microsoft o Human Computer Resources distribuyen versiones adaptadas a ordenadores más modestos.

Todas eran versiones plenamente funcionales pero carecían de un soporte de red adecuado. Los experimentos con Ethernet no habían salido del todo bien y lo único que existía era un servicio muy pobre que permitía compartir datos y reenviar correo: el UUCP (Unix to Unix Copy Program), que había sido desarrollado por los Bell Labs para distribuir software a través de modems conectados a la línea telefónica convencional.

El departamento de defensa estadounidense acababa de financiar el desarrollo de la pila de protocolos TCP/IP y necesitaba un equipo para implementarla en su red experimental ARPANET. El equipo de la universidad de Berkeley estaba en el momento adecuado, en el sitio adecuado y fue elegido para llevar a cabo esta tarea. Este hecho es probablemente el punto de inflexión más grande que ha experimentado Unix en toda su historia, ya que contar con una implementación de TCP/IP dotaba al sistema operativo de la funcionalidad que le faltaba y que tanto éxito le ha supuesto: la conectividad. Con los sockets de Berkeley, Unix contaba con una interfaz sencilla que permitía enviar de forma fiable información desde un ordenador a otro, incluso cuando ambos estaban conectados a diferentes redes. Esto supone el *boom* definitivo de Unix, que se alza entonces como el sistema operativo omnipresente en todos servidores de red y supercomputadoras del mundo.

Ya en la década de los 80 son dos los hechos que marcan negativamente la trayectoria de Unix.

Por una parte, en 1981, Bill Gates compra a Seattle Computer Products el QDOS (Quick and Dirty Operating System) por 50.000 dólares, lo que resulta ser una de las mejores inversiones de la historia ya que los beneficios que le reporta la comercialización de un software no escrito por él y rebautizado como MS-DOS son del orden de billones de dólares. Microsoft se hace así con el control de todo el mercado de los ordenadores personales que, al no contar con el hardware adecuado para hacer funcionar Unix, tenían que conformarse con sistemas operativos de categoría muy inferior, como el suministrado por Gates.

Por otra parte, surgen batallas legales entre la universidad de Berkeley y la AT&T. Al comprar la licencia de la AT&T, la universidad quería experimentar con aquel sistema operativo moderno que incluía código fuente. El equipo de investigación escribía decenas de utilidades nuevas y las redistribuía con el nombre de BSD (Berkeley Software Distribution). En principio todos los cambios eran distribuidos a poseedores de una licencia de la AT&T pero llega un momento en que la universidad cambia tantos aspectos internos del sistema operativo que la

diferencia entre las dos versiones se hace enorme y Berkeley considera que puede redistribuirlo como propio a cualquier persona que no posea la licencia original. Esto enfurece a la AT&T quien demanda a Berkeley por infracción de copyright, quien a su vez demanda a AT&T por copiar fragmentos de código. Comienza entonces una enorme batalla judicial en la que ambas partes se acusan mutuamente y que se prolonga hasta 1994 cuando Novell, que dos años antes había adquirido todos los derechos del Unix de la AT&T, firma la paz con la universidad de California. El acuerdo adoptado consistía en que Berkeley debía eliminar todo el código residual de AT&T y lanzar una última distribución de BSD totalmente libre de problemas de licenciamiento. Esta distribución fue el 4.4-BSD Lite2.

En mitad de toda esta batalla legal un estudiante finlandés llamado Linus Torvalds, enfadado por los altos precios que tenían los Unix para x86, decide crear su propio sistema operativo inspirándose en Minix, un sistema creado por el profesor Andrew Tanenbamum como recurso pedagógico.

En cuanto consigue un prototipo funcional publica un mensaje en Usenet:

¿Suspiras por los fabulosos días de Minix-1.1, cuando los hombres eran hombres y escribían sus propios controladores de dispositivo? ¿Estás sin ningún proyecto decente y te mueres por hincarle los dientes a un sistema operativo que puedas modificar para tus necesidades? ¿Te parece frustrante que todo en Minix funcione? Se te acabaron las noches en vela tratando de conseguir que un programa funcionase? Entonces este post puede ser para ti ;-)

Como dije hace un mes, estoy trabajando en una versión libre de una especie de Minix para ordenadores AT-386. Por fin ha llegado a una fase en
la que se puede usar (o no, dependiendo de lo que quieras) y me gustaría publicar el código fuente para que se extienda. Solo es la versión 0.02 pero he conseguido ejecutar en él bash/gcc/gnu-make/gnu-sed/compress etc.
[...]

El mensaje suscitó un enorme interés en un sector de la comunidad internauta y fueron muchos los programadores que decidieron unirse a este proyecto, que adoptó el nombre de Linux en honor a su creador. Torvalds solo estaba interesado en diseñar el núcleo del sistema operativo por lo que, para conseguir un producto completo, se necesitaba implementar todo el conjunto de herramientas que tradicionalmente habían acompañado a los sistemas UNIX. El proyecto GNU de Richard Stallman encajaba como anillo al dedo ya que éste había estado años desarrollando un clon libre de UNIX y ya contaba con todo el conjunto de herramientas necesarias. Curiosamente a Stallman le faltaba implementar el kernel del sistema por lo que la unión fue perfecta y así surgió GNU/Linux. A partir de ese momento GNU/Linux no ha hecho más que extenderse a una velocidad vertiginosa, con millones de usuarios en el mundo entero. Este hecho ha sido probablemente el segundo más importante en la historia del sistema UNIX ya que su filosofía de código abierto ha conseguido extender el sistema operativo, y en general el software libre, incluso hasta los ordenadores de sobremesa, terreno que siempre había estado copado al 100% por Microsoft y Apple.

En la actualidad los diversos sabores de Unix son el sistema utilizado en la mayoría de los sistemas de cálculo y experimentación científica, de servidores de red, de control

de procesos industriales y de supercomputadores en general. Gracias a Linux también posee una cuota nada despreciable en el mercado de los ordenadores de sobremesa y se prevé un crecimiento exponencial, pudiendo a corto-medio plazo llegar a convertirse en un serio competidor a los sistemas operativos de la compañía Microsoft.

FILOSOFÍA

Unix ha probado lo que ningún otro sistema operativo puede imaginar. Diferentes versiones han funcionado sobre supercomputadores, PDAs, hardware de red, teléfonos móviles, videoconsolas etc, pero Unix es mucho más un sistema operativo. Unix se caracteriza por tener una filosofía y una forma diferente de ver las cosas, así como por ser el sistema operativo de los entusiastas de la informática. La complejidad de uso que le achacan los defensores de sistemas inferiores como MS Windows es vista por los usuarios de Unix como uno de sus mayores atractivos. La idea de conocer el funcionamiento interno del sistema, la posibilidad de modificar sus partes, de añadir nueva funcionalidad o su enorme fiabilidad y ausencia de cuelgues totales, ha hecho que mucha gente se identifique totalmente con este sistema operativo y se convierta en un ferviente defensor del mismo.

Al igual que Internet, Unix tiene una cultura propia, tiene un estilo de programación característico y lleva consigo una potente filosofía de diseño. Entender el mundo y la comunidad que rodea a UNIX es esencial para poder desarrollar software para él. Frecuentemente se acusa a esta comunidad de ser un grupo muy cerrado y reticente al cambio. Esto es bastante cierto y no parece que vaya a cambiar en un futuro próximo por lo que es conveniente familiarizarse con la filosofía que rodea al sistema para poder integrarse en su mundo.

Básicamente toda la filosofía de la programación en Unix se puede resumir con la frase "*Do one thing and do it well*", pero en general hay una serie de pautas que deberían tenerse en cuenta:

-> Haz que cada programa haga una cosa y la haga bien.
-> Para llevar a cabo una nueva tarea escribe un programa nuevo. No compliques uno viejo añadiendo nueva funcionalidad.
-> Escribe tu programa teniendo en cuenta que su salida probablemente sea la entrada de otro programa. No llenes stdout con información innecesaria ni utilices formatos raros.
-> Guarda los datos en archivos de texto plano. Si necesitas seguridad, confía en los permisos.
-> Usa nombres cortos y en minúscula.
-> Si no es imprescindible, no pidas nada de forma interactiva: haz que el usuario suministre los datos por línea de comandos en la llamada.
-> Haz partes simples conectadas mediante interfaces limpias y bien definidas.
-> Céntrate en los datos. Si has elegido las estructuras adecuadas y organizado todo correctamente, los algoritmos serán evidentes.
-> Claridad mejor que complejidad. La solución más simple es frecuentemente la mejor: añade complejidad solo donde sea indispensable.
-> Portabilidad mejor que eficiencia.
-> Piensa en paralelo. Hay otros procesos en el mundo, incluso instancias de tu mismo

programa funcionando al mismo tiempo.
-> Haz un programa grande sólo cuando haya quedado demostrado que no puede hacerse con uno pequeño.
-> Si tu programa no tiene nada interesante que decir, que no diga nada.
-> No existe una única manera de hacerlo. Cada problema tiene múltiples soluciones.
-> Diseña pensando en el futuro, está más cerca de lo que piensas.
-> Unix no pide por favor.

En caso de duda la regla universal a tener en cuenta es siempre la norma KISS: *Keep it simple, stupid!*

ESTÁNDARES

En 1973 el código fuente de Unix se reescribía en el recién creado lenguaje C. Esto suponía no tener que preocuparse de las peculiaridades del procesador de la máquina subyacente por lo que se hacía muy sencillo modificar el sistema operativo o portarlo a otras arquitecturas. Como consecuencia, el desarrollo de Unix se divide en varias ramas que se comienzan a alejar en poco tiempo. AT&T introducía su Unix System 3, la universidad de Berkeley desarrollaba el BSD v4 y algunos vendedores como Interactive Systems, Microsoft o Human Computer Resources distribuían versiones adaptadas a ordenadores más modestos. Surgió entonces la necesidad de crear unas pautas generales que permitieran que los distintos sabores de Unix que habían surgido fuesen compatibles entre sí. Es lo que más tarde se conocerá como el estándar POSIX.

Una organización llamada originalmente /usr/group decidió, en 1980, ponerse manos a la obra y formar un comité para redactar un documento que especificase claramente la interfaz entre el sistema operativo y el programador. Para hacer la tarea más fluida se decidió limitar a 40 el número de personas que formaran el comité y se estableció que las decisiones fueran aprobadas por al menos 2/3 de los participantes. También se fijaron dos puntos fundamentales a tener en cuenta en todo el desarrollo del documento:

1- Que el documento resultante fuese independiente de cualquier versión de Unix ya desarrollada, permitiendo así que futuras empresas pudiesen crear aplicaciones o incluso sistemas operativos en base a las especificaciones, sin tener que comprar los productos de un solo distribuidor.

2- Que el proceso se centrase en la definición clara y no ambigua de la interfaz entre el sistema operativo y el usuario, dejando aparte temas como la administración, las comunicaciones o las shell del sistema.

El proceso no fue fácil ni corto. Cada decisión debía meditarse seriamente considerando lo que ya existía y lo que debería o no existir, sin centrarse en las versionas más extendidas y sin dejar de lado las menos importantes. Se tardó cuatro años en redactar un borrador del documento. Dicho borrador fue enviado al

comité de estándares del IEEE quien puso a trabajar en el asunto a su Comité Técnico de Sistemas operativos: el TCOS-SS. En abril de 1986 el IEEE hizo pública la versión de prueba del estándar a fin de conocer las opiniones de las diversas empresas y profesionales del sector. Dos años más tarde, en agosto de 1988, se publicó la primera versión definitiva: el IEEE 1003.1-1988, más comúnmente conocida como POSIX.1, que quedó ratificada dos años más tarde, al estandarizarse también el lenguaje de programación C.

Aun así, un solo documento no era suficiente por lo que el IEEE siguió con su tarea. En 1990 ya había 10 proyectos aprobados y unas 300 personas participaban en jornadas intensivas de una semana, una vez al mes. Se empezó a trabajar en la estandarización de los comandos y utilidades del sistema operativo (POSIX.2) para seguir con metodologías de testeo (POSIX.3), aplicaciones en tiempo real (POSIX.4), las interfaces entre el sistema operativo y los lenguajes ADA (POSIX.5) y FORTRAN 77 (POSIX.9), súper computación (POSIX.10), etc.

Debe quedar claro que el estándar solo define la interfaz, en ningún caso la implementación y que está pensado para ser utilizado tanto por desarrolladores de aplicaciones como por programadores que implementen sistemas operativos. POSIX es por tanto una colección de documentos que definen claramente una interfaz estándar entre el sistema operativo y sus aplicaciones a nivel de código fuente. En otras palabras, POSIX define los servicios que debe proveer un sistema, definiendo de forma exacta los prototipos de las funciones de biblioteca y llamadas al sistema, los tipos de las variables utilizadas, las cabeceras, los códigos de retorno de las funciones, su comportamiento concurrente, etc. Además, tambien especifica otros aspectos como por ejemplo la estructura general del sistema de ficheros, consideraciones sobre el set de caracteres usado, sobre las expresiones regulares, las variables del entorno, la interacción con el terminal o las secuencias de escape.

En la actualidad, la rama POSIX más importante es sin duda la POSIX.1x, basada en el popular lenguaje C. A continuación se resumen de forma general las pautas definidas en el documento.

- POSIX.1, Servicios centrales. (incorpora el estándar ANSI C)
    - Creación y control de procesos
    - Señales
    - Excepciones en operaciones de coma flotante
    - Violaciones de segmento
    - Instrucciones ilegales
    - Errores de Bus
    - Temporizadores
    - Operaciones con ficheros y directorios
    - Tuberías
    - La librería estándar de C
    - Interfaces para el control de los puertos de entrada/salida.

- POSIX.1b, Extensiones para el procesado en tiempo real
    - Planificacion por prioridades
    - Señales en tiempo real
    - Relojes y temporizadores
    - Semáforos
    - Paso de mensajes

    - Memoria compartida
    - Entrada/salida síncrona
    - Entrada/salida asíncrona
    - Protección de áreas de memoria.
- POSIX.1c, Extensiones multihilo
    - Creación, control y limpieza de hilos.
    - Planificación de hilos
    - Sincronización de hilos
    - Manejo de señales.
- POSIX.1d Extensiones adicionales para el procesamiento en tiempo real
- POSIX.1j Extensiones avanzadas para el procesamiento en tiempo real
- POSIX.1q Tracing:
    - Recolección y presentación de logs de las llamadas al sistema
    - Recolección y presentación de la actividad de entrada/salida
    - Recolección y presentación de eventos definidos por el usuario.

BIBLIOGRAFÍA

The Art of Unix Programming - Eric Steven Raymond [web]
UNIX - Artículo de la Wikipedia [web]
A Brief History of Unix - Charles Severance [web]
History of UNIX / Linux and other variants - Computer Hope [web]
History of Unix - Ronda Hauben [web]
The POSIX Family of Standards - Stephen R. Walli [pdf]
POSIX - Artículo de la Wikipedia [web]
Unix philosophy - Artículo de la Wikipedia [web]
Standards-the history of Posix: a study in the standards process - Isaak, J.
POSIX: A Case Study in a Successful Standard - Stephen R. Walli [doc]

REFERENCIAS

The UNIX Time-Sharing System - Dennis Ritchie and Ken Thompson **-** Communications of the ACM, 17, No. 7 (July 1974). [web]
IEEE Std 1003.1 [web]
comp.os.minix > Free minix-like kernel sources for 386-AT - Linus Torvalds [web]

ENLACES INTERESANTES:

LOS DIVERSOS SABORES

FreeBSD [web]
OpenBSD [web]
NetBSD [web]
Linux Kernel [web]
GNU Project [web]
Red Hat Linux [web]
SUSE Linux [web]
Debian GNU/Linux [web]
SCO UnixWare [web]
Sun Solaris [web]
HP-UX [web]
IBM AIX [web]
SG IRIX [web]

LOS PROTAGONISTAS

AT&T [web]
Novell [web]
Universidad de California, Berkeley [web]
Brian Kernighan [web personal] [biografia]
Dennis Ritchie [web personal] [biografia]
Ken Thompson [web personal] [biografia]
Linus Torvalds [web personal] [biografia]
Richard Stallman [web personal] [biografia]
Eric S. Raymond [web personal] [biografia]

LIBROS

Magic Garden Explained **-** Berny Goodheart [Amazon]
The Art of Unix Programming **-** Eric S. Raymond [Amazon]
Advanced Programming in the UNIX(R) Environment - W. Richard Stevens [Amazon]
Unix Network Programming - W. Richard Stevens [Amazon]
POSIX Programmer's Guide **-** Donald Lewine [Amazon]

UNIX Internals: The New Frontiers **-** Uresh Vahalia [Amazon]
Bsd Kernel Internals - Nathan Boeger [Amazon]
Solaris Internals: Architecture and Techniques **-** James Mauro [Amazon]
Linux Kernel Development - Robert Love [Amazon]
A Quarter Century of Unix - Peter H. Salus [Amazon]
The Cathedral & the Bazaar - Eric S. Raymond [Amazon]

OTROS

Linea del tiempo de Unix [web]
The Unix Haters Handbook [wikipedia] [libro]
Microsoft Linux [web]
/~roth/geek-humor/unix [web]
Politically Correct UNIX [web]

APÉNDICE ;-)

EL TREN
Tres ingenieros de la Universidad de Berkeley que desarrollaban el BSD 4 y tres de Microsoft que trabajaban en el MS-DOS 2.0 se disponían a viajar en tren para asistir a un congreso. En la estación, los tres de Microsoft compraron sus respectivos billetes y vieron cómo los ingenieros de Berkeley sólo compraban un billete…

- "¿Cómo van a viajar tres personas con un solo billete?", les preguntó uno de los empleados de Microsoft.
- "Mira y verás!", le respondió uno de los BSDeros.

Total, se subieron todos ellos al tren… Los empleados de Microsoft tomaron sus respectivos asientos y vieron cómo los ingenieros de Berkeley se metían los tres en el aseo, cerrando la puerta. Al poco de arrancar el tren, llegó el revisor pidiendo los billetes, tocó en la puerta del aseo y dijo: "billete por favor"… La puerta se abrió lo suficiente como para que saliese un brazo con el billete en la mano, el revisor lo picó, lo devolvió y se marchó… Al ver esto, los empleados de Microsoft pensaron que era una idea genial, y que por lo tanto, para no quedarse fuera de juego, copiarían el truco a la vuelta del congreso, para de esa manera ahorrarse un dinerillo y demostrarle al jefe Bill Gates lo inteligentes que habían sido.

A la vuelta, en la estación, los empleados de Microsoft sacaron un solo billete, quedándose atónitos al ver que los de BSD no sacaban ninguno…

- "¿Como vais a viajar sin billetes?", pregunto perplejo uno de los empleados de Microsoft.
- "Mira y verás!", le respondió uno de los BSDeros.

Al subir al tren, los tres empleados de Microsoft se metieron en un aseo y los tres ingenieros de Berkeley en otro… Arrancó el tren, y rápidamente uno de los BSDeros salió de su aseo, se dirigió al aseo de los empleados de Microsoft, tocó en la puerta y dijo: "billete, por favor"…

LA CLASE

Se cuenta por ahí que un profesor un módulo de FP en Desarrollo de Aplicaciones explicaba en clase que los buenos programadores solo usan Windows XP y pide que levante la mano todo el que también sea seguidor de la empresa de Bill Gates. Todos en clase, por temor a represalias por parte del profesor, levantan la mano, excepto uno con pinta de friki que estaba sentado al fondo del aula. El profesor le miró con sorpresa y le preguntó:

-Oiga, usted. ¿Por qué no ha levantado la mano?.
-Porque yo no utilizo Windows.

El profesor, extrañado, preguntó de nuevo:

-Vaya, y sino utilizas Windows ¿Que Sistema Operativo utilizas?
-GNU/Linux. -Respondió orgulloso-

El profesor, cuyos fanáticos oídos no podían dar crédito a algo así, exclamó:

-Pero hijo mío ¿qué pecado has cometido para utilizar tal chapuza?

El alumno, muy tranquilo, le respondió:

-Mi padre es informático y usa SUSE Linux, mi madre es asesora en seguridad y usa Debian Linux y mi hermano estudia Físicas y utiliza Linux Mandrake, por eso yo también utilizo GNU/Linux! -remató orgulloso y convencido-

- Bueno, -replicó irritado el profesor-, pero ese no es motivo para utilizar Linux. Tú no tienes porqué hacer lo que hacen tus padres.. Por ejemplo, si tu madre se prostituyese y se drogase todo el día, tu padre se tocara los cojones, bebiese como un cabrón y traficase con drogas y tu hermano atracase comercios y robase a abuelitas, entonces, ¿tú qué harías?

- Seguramente instalaría Windows...