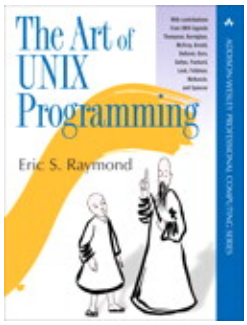
RevisiÃ³n bibliogrÃ¡fica: El Arte de Programar en Unix.



by Edgar HernÃ¡ndez ZÃºÃ±iga.
<edgar(en)linuxfocus.org>

About the author:

AquÃ- deberÃ-a escribir un poco de mi pero no hay tiempo para ello.

Abstract:

La presente revisiÃ³n bibliogrÃ¡fica tratarÃ¡ de mostrar una perspectiva clara y concisa de los temas centrales del libro que para la fecha en la que se lee esto, debe estar listo para su venta.

La revisiÃ³n se basÃ³ en la versiÃ³n 0.87 del libro, ejemplar previo a la publicaciÃ³n que se me hizo llegar para evaluarlo antes de su publicaciÃ³n, en el transcurso de la realizaciÃ³n de este artÃ-culo me di cuenta de que la amplitud del tema que trata, aunado a la calidad de la manera en la que lo hace, quizÃ¡ merecerÃ-a un artÃ-culo mÃ¡s para continuar exponiendo ampliamente las caracterÃ-sticas del libro, espero que sea asÃ-.

TÃ-tulo revisado:	El Arte de Programar en UNIX.
TÃ-tulo original:	The Art of UNIX Programming.
Autor(es):	Eric S. Raymond.
Colaboraciones:	Thompson, Kernighan, McIlroy, Arnold, Bellovin, Korn, Gettys, Packard, Lesk, Feldman, McKusick, Spencer.
PÃ¡ginas:	550 en esta versiÃ³n.

IntroducciÃ³n

Eric S. Raymond, conocido por todos por La Catedral y El Bazar, realiza un trabajo excepcional. Este libro serÃ¡ publicado en breve pero puede encontrarse en lÃ-nea y es, de hecho, con base en esa gran recopilaciÃ³n

de informaciÃ³n basada en horas y horas de investigaciÃ³n y procesos que se nos permite tener una visiÃ³n general de muchas de las tecnologÃ-as y elementos que conforman a los sistemas Unix y a algunos de sus derivados, por la que se ha conseguido llevar a la imprenta este libro.

El trabajo de Eric S. Raymond se ve apoyado de gran manera por la colaboraciÃ³n de grandes nombres, entre ellos Ken Thompson, Brian Kernighan y Dennis Ritchie; curiosamente son el primero y Ãºltimo de quienes acepta haber recibido la motivaciÃ³n para realizar este trabajo.

El libro se encuentra dividido en cuatro partes esenciales:

- Contexto
- DiseÃ±o
- ImplementaciÃ³n
- Comunidad

Cada una de estas partes contiene diversos temas que van desde *Conceptos bÃ¡sicos de la filosofÃ-a Unix* perteneciente al Contexto, hasta *Mejores prÃ¡cticas para trabajar con desarrolladores de CÃ³digo Abierto*, pasando por conceptos de *Modularidad, DiseÃ±o de protocolos para aplicaciones, Transparencia, Minilenguajes y Complejidad* en DiseÃ±o y *Lenguajes y Herramientas* en la parte que corresponde a la ImplementaciÃ³n. AdemÃ¡s de todo esto, cada vez que se hace necesario se muestra un ejemplo prÃ¡ctico que permite al lector incrementar la compresiÃ³n de la lectura.

Muy a mi pesar, ya que disfruto mÃ¡s de hacer una revisiÃ³n profunda de los libros que solamente citar el contenido y dar una breve opiniÃ³n, pero teniendo en cuenta que para los lectores pueda ser Ãºtil, aÃ±adÃ- el Ã-ndice general del libro. Nuestro siguiente artÃ-culo serÃ¡ de un anÃ¡lisis mÃ¡s profundo del libro.

Tabla de contenido

Elementos de estilo del sistema operativo.
Comparaciones de sistemas operativos.
QuÃ© hay alrededor.

II. DISEÃ‘O.

4. Modularidad.

EncapsulaciÃ³n y tamaÃ±o Ã³ptimo de los mÃ³dulos.
CompactaciÃ³n y ortogonalidad.
Bibliotecas.
Unix y los lenguajes orientados a objetos.
CodificaciÃ³n para la modularidad.

5. Textualidad.
La importancia de ser textual.
Metaformatos para archivos de datos.
DiseÃ±o de protocolos de aplicaciÃ³n.
Metaformatos de protocolo de aplicaciÃ³n.

6. Transparencia.
Algunos casos de estudio.
DiseÃ±ando para la transparencia y el descubrimiento.
DiseÃ±ando para la sustentabilidad.

7. ProgramaciÃ³n mÃºltiple.
Control de separaciÃ³n de la complejidad para ajustar el rendimiento.
TaxonomÃ-a de los mÃ©todos IPC en Unix.
Problemas y metÃ³dos a evitar.
Procesos de particionamiento a nivel diseÃ±o.

8. Minilenguajes.
TaxonomÃ-a de los lenguajes.
Aplicando minilenguajes.
DiseÃ±ando minilenguajes.

9. TransformaciÃ³n.
ProgramaciÃ³n de control de datos.
Generando cÃ³digo ad−hoc.

10. ConfiguraciÃ³n.
Â¿QuÃ© deberÃ-a ser configurable?
DÃ³nde estÃ¡n las configuraciones.
Archivos de control de arranque.
Variables de ambiente.
Opciones de la interfaz de comandos.
CÃ³mo escoger entre los mÃ©todos de configuraciÃ³n.
Al romper estÃ¡s reglas.

11. Interfaces.
AplicaciÃ³n de la regla: La menor sorpresa.
Historia del diseÃ±o de la interfaz Unix.

Evaluando diseÃ±os de interfaz.
Compensaciones entre CLI y las interfaces visuales.
Transparencia, expresividad, y configurabilidad.
Patrones de diseÃ±o de la interfaz Unix.
Aplicando los patrones de diseÃ±o de la interfaz Unix.
El navegador web como front end universal.
El silencio es de oro.

12. OptimizaciÃ³n.
No solamente hagas algo, Â¡mantÃ©nlo!.
Medir antes de optimizar.
Lo no adecuado es considerado daÃ±ino.
Rendimiento de procesamiento contra latencia.

13. Complejidad.
Hablando de complejidad.
Una historia de cinco editores..
El tamaÃ±o indicado para un editor.
El tamaÃ±o indicado del software.

III. IMPLEMENTACIÃ“N.

14. Lenguajes.

Cornucopia de lenguajes Unix.
C, Â¿por quÃ© no?
Lenguajes interpretados y estrategias mixtas.
Evaluaciones de lenguajes.
Tendencias para el futuro.
Escogiendo un conjunto de herramientas para X.

15. Herramientas.
Un sistema operativo amistoso con el desarrollador.
Escogiendo un editor.
Generadores de cÃ³digo para propÃ³sitos especiales.
Usando make en otras partes ademÃ¡s de C/C++.
Sistemas de control de versiones.
DepuraciÃ³n durante el tiempo de ejecuciÃ³n.
Perfilar.
Emacs como front end universal.

16. Reuso.
La historia de J. Random Newbie.
Transparencia como llave del reuso.
Del reuso al CÃ³digo Abierto.
Las mejores cosas de la vida son abiertas.
Â¿En dÃ³nde debo mirar?.
Â¿CuÃ¡les son los temas en el uso de software de CÃ³digo Abierto?
Temas de licenciamiento.

IV. COMUNIDAD.

17. Portabilidad.

Cornucopia de lenguajes Unix.
EvoluciÃ³n de C.
EstÃ¡ndares Unix.
Especificaciones como DNA, cÃ³digo comp RNA.
Programando para la Portabilidad.
InternacionalizaciÃ³n.
Portabilidad, estÃ¡ndares abiertos y CÃ³digo Abierto.

18. DocumentaciÃ³n.
Conceptos de documentaciÃ³n.
El estilo Unix.
El zoolÃ³gico de los formatos de documentaciÃ³n Unix.
El caos actual y las posibles opciones.
La herramienta DocBook.
CÃ³mo escribir documentaciÃ³n Unix.

19. CÃ³digo Abierto.
Unix y CÃ³digo Abierto.
Mejores prÃ¡cticas para trabajar con desarrolladores de CÃ³digo Abierto.
La lÃ³gica de las licencias: CÃ³mo escoger una.
PorquÃ© debes usar una licencia estÃ¡ndar.
Variedades en el licenciamiento de CÃ³digo Abierto.

20. A futuro.
Esencia y accidente en la tradiciÃ³n Unix.
Problemas en el diseÃ±o de Unix.
Problemas en el ambiente de Unix.
Problemas en la cultura de Unix.
Razones para creer.

A. Glosario de abreviaciones.
B. Referencias.
C. Contribuidores.

Cultura y filosofÃ-a Unix

Como mencionaba antes, ademÃ¡s de un opiniÃ³n personal acerca del libro, creo que se puede tomar una amplia ventaja de realizar una anÃ¡lisis sistemÃ¡tico que permita mostrar tanto el contenido del libro como una idea clara del mismo para aquellos que quizÃ¡ no podrÃ¡n leerlo completamente.

Para quienes cuentan con experiencia en el mundo Unix, asi como algunos sistemas operativos derivados en mayor o menor grado de este, no es nada raro escuchar el tÃ©rmino: *FilosofÃ-a*, Unix es una filosofÃ-a, ademÃ¡s de un cultura es un modo de vida, un modo de hacer las cosas, una manera en la cual se consigue un esquema completamente distintivo de programaciÃ³n, por ejemplo.

Unix estÃ¡ basado en una poderosa filosofÃ-a de diseÃ±o que le ha permitido, desde su nacimiento en 1969, ser una referencia para la creaciÃ³n de gran cantidad de sistemas operativos.

Lo bÃ¡sico de la filosofÃ-a Unix

La filosofÃ-a Unix, iniciada evidentemente por Ken Thompson entre su confusiÃ³n por diseÃ±ar un sistema operativo pequeÃ±o pero altamente capaz y sus lecciones aprendidas de diversas fuentes, no puede determinarse como un mÃ©todo formal de diseÃ±o, estÃ¡ basada completamente en la experiencia.

En el libro encontrarÃ¡s grandes temas para estudio, una de las partes de las cuales se puede tomar provecho es aquella en la que se nos motiva a abstraernos a fin de mantener en mente algunas de las siguientes ideas bÃ¡sicas que forman parte de la filosofÃ-a Unix:

Modularidad: Escribir partes de cÃ³digo pequeÃ±as pensando en conectarlas a travÃ©s de interfaces.
ComposiciÃ³n: DiseÃ±ar programas que se conectarÃ¡n a otros programas.
EconomÃ-a: El tiempo del programador es caro, consÃ©rvalo.
OptimizaciÃ³n: Realiza un prototipo antes de pulir. Ponlo a trabajar antes de que pienses en optimizarlo.
Extensibilidad: DiseÃ±a para el futuro, porque llegarÃ¡ antes de lo que te imaginas.

ConclusiÃ³n y recomendaciones

Es, y sin temor a equivocarme un libro excelente. Eric Steven Raymond, hace un gran trabajo de recopilaciÃ³n y con un estilo verdaderamente bueno aÃ±adido a las referencias usadas para la creaciÃ³n de este libro, como: *El entorno de programaciÃ³n Unix* de Kernighan y Pike, *La filosofÃ-a Unix* de Gancarz y, *El programador pragmÃ¡tico* de Hunt y Thomas entre otros, consigue hacer necesaria su lectura, el libro estÃ¡ dirigido a todos aquellos programadores que novatos, desean adquirir la experiencia necesario para mejorar, asÃ- como para programadores que ajenos a Unix conocen de lenguajes como C, C++ y Java.

Tiene una bibliografÃ-a extensa que permite ampliar horizontes si asÃ- se desea y un tema que en particular disfrutÃ© fue el de estÃ¡ndares para la creaciÃ³n de documentaciÃ³n y sistemas de control de versiones.

Espero generar un artÃ-culo que continÃºe con este para hacer un anÃ¡lisis mÃ¡s profundo del libro, mientras, me despido esperando que hayan disfrutado esta breve introducciÃ³n.

2005−01−10, generated by lfparser_pdf version 2.51