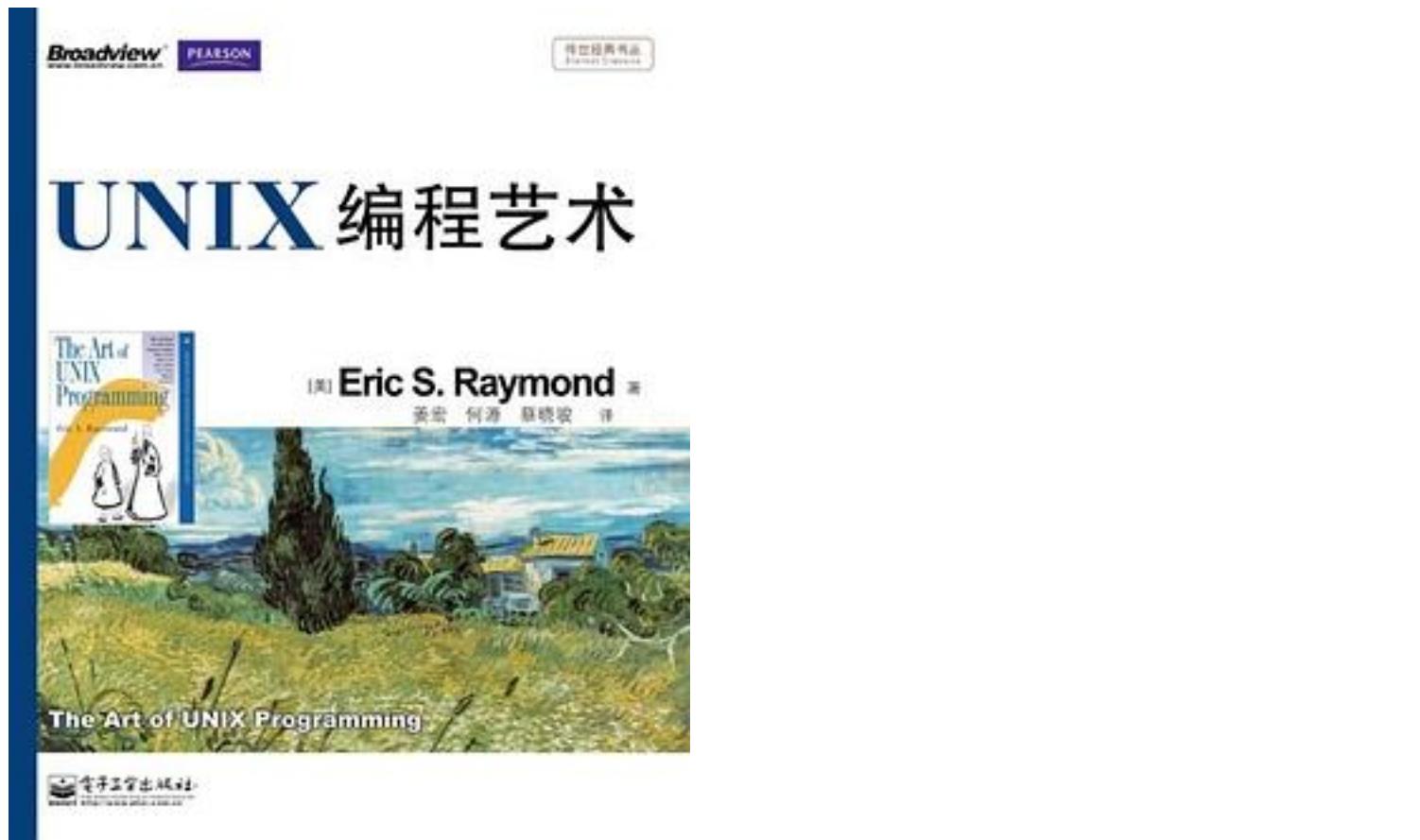


# UNIX编程艺术



[UNIX编程艺术 下载链接1](#)

著者:[美] Eric S · Raymond

出版者:电子工业出版社

出版时间:2011-1

装帧:

isbn:9787121123290

本书主要介绍了Unix系统领域中的设计和开发哲学、思想文化体系、原则与经验，由公认的Unix编程大师、开源运动领袖人物之一Eric S. Raymond倾力多年写作而成。包括Unix设计者在内的多位领域专家也为本书贡献了宝贵的内容。本书内容涉及社群文化、软件开发设计与实现，覆盖面广、内容深邃，完全展现了作者极其深厚的经验积累和领域智慧。

## 作者介绍:

《The Art of UNIX Programming》，简称TAOUP，作者Eric S. Raymond，简称ESR。这大概是计算机类书籍中很少见的一本课外读物。TCP/IP编程之类典型Unix编程书中讲到的东西在这本书里面找不到，所以书里讲到的当然就是别的书里找不到的东西。读者也许需要有相当的Unix背景、或者长期钻研某个专题，才能体会到作者的弦外之音。ESR作为老牌黑客信手拈来的典故，如果不是在Unix里面长期浸淫，大概很难有所共鸣，所以把这当作Unix的一部坊间史话倒也合适。

## 目录: 序 xxv

### part i 1

#### 第1章 哲学 3

1.1 文化？什么文化 3

1.2 unix的生命力 4

1.3 反对学习unix文化理由 5

1.4 unix之失 6

1.5 unix之得 7

1.5.1 开源软件 7

1.5.2 跨平台可移植性和开放标准 8

1.5.3 internet和万维网 8

1.5.4 开源社区 9

1.5.5 从头到脚的灵活性 9

1.5.6 unix hack之趣 10

1.5.7 unix的经验别处也可适用 11

1.6 unix哲学基础 11

1.6.1 模块原则：使用简洁的接口拼合简单的部件 14

1.6.2 清晰原则：清晰胜于机巧 14

1.6.3 组合原则：设计时考虑拼接组合 15

1.6.4 分离原则：策略同机制分离，接口同引擎分离 16

1.6.5 简洁原则：设计要简洁，复杂度能低则低 17

1.6.6 吝啬原则：除非确无它法，不要编写庞大的程序 18

1.6.7 透明性原则：设计要可见，以便审查和调试 18

1.6.8 健壮原则：健壮源于透明与简洁 18

1.6.9 表示原则：把知识叠入数据以求逻辑质朴而健壮 19

1.6.10 通俗原则：接口设计避免标新立异 20

1.6.11 缄默原则：如果一个程序没什么好说的，就保持沉默 20

1.6.12 补救原则：出现异常时，马上退出并给出足量错误信息 21

1.6.13 经济原则：宁花机器一分，不花程序员一秒 22

1.6.14 生成原则：避免手工hack，尽量编写程序去生成程序 22

1.6.15 优化原则：雕琢前先得有原型，跑之前先学会走 23

1.6.16 多样原则：决不相信所谓“不二法门”的断言 24

1.6.17 扩展原则：设计着眼未来，未来总比预想快 24

1.7 unix哲学之一言以蔽之 25

1.8 应用unix哲学 26

1.9 态度也要紧 26

#### 第2章 历史——双流记 29

2.1 unix的起源及历史，1969—1995 29

2.1.1 创世纪：1969—1971 30

2.1.2 出埃及记：1971—1980 32

2.1.3 tcp/ip 和 unix内战：1980—1990 35

2.1.4 反击帝国：1991—1995 41

2.2 黑客的起源和历史：1961—1995	43
2.2.1 游戏在校园的林间：1961—1980	44
2.2.2 互联网大融合与自由软件运动：1981—1991	45
2.2.3 linux 和实用主义者的应对：1991—1998	48
2.3 开源运动：1998年及之后	49
2.4 unix的历史教训	51
第3章 对比： unix哲学同其他哲学的比较	53
3.1 操作系统的风格元素	53
3.1.1 什么是操作系统的统一性理念	54
3.1.2 多任务能力	54
3.1.3 协作进程	55
3.1.4 内部边界	57
3.1.5 文件属性和记录结构	57
3.1.6 二进制文件格式	58
3.1.7 首选用户界面风格	58
3.1.8 目标受众	59
3.1.9 开发的门坎	60
3.2 操作系统的比较	61
3.2.1 vms	61
3.2.2 macos	64
3.2.3 os/2	65
3.2.4 windows nt	68
3.2.5 beos	71
3.2.6 mvs	72
3.2.7 vm/cms	74
3.2.8 linux	76
3.3 种什么籽，得什么果	78
part ii	81
第4章 模块性：保持清晰，保持简洁	83
4.1 封装和最佳模块大小	85
4.2 紧凑性和正交性	87
4.2.1 紧凑性	87
4.2.2 正交性	89
4.2.3 spot原则	91
4.2.4 紧凑性和强单一中心	92
4.2.5 分离的价值	94
4.3 软件是多层的	95
4.3.1 自顶向下和自底向上	95
4.3.2 胶合层	97
4.3.3 实例分析：被视为薄胶合层的c语言	98
4.4 程序库	99
4.4.1 实例分析：gimp插件	100
4.5 unix和面向对象语言	101
4.6 模块式编码	103
第5章 文本化：好协议产生好实践	105
5.1 文本化的重要性	107
5.1.1 实例分析：unix口令文件格式	109
5.1.2 实例分析：.newsrc格式	110
5.1.3 实例分析：png图形文件格式	111
5.2 数据文件元格式	112
5.2.1 dsv 风格	113
5.2.2 rfc 822 格式	114
5.2.3 cookie-jar格式	115
5.2.4 record-jar格式	116

5.2.5 xml	117
5.2.6 windows ini 格式	119
5.2.7 unix文本文件格式的约定	120
5.2.8 文件压缩的利弊	122
5.3 应用协议设计	123
5.3.1 实例分析: smtp, 一个简单的套接字协议	124
5.3.2 实例分析: pop3, 邮局协议	124
5.3.3 实例分析: imap, 互联网消息访问协议	126
5.4 应用协议元格式	127
5.4.1 经典的互联网应用元协议	127
5.4.2 作为通用应用协议的http	128
5.4.3 beep: 块可扩展交换协议	130
5.4.4 xml-rpc, soap和jabber	131
第6章 透明性: 来点儿光	133
6.1 研究实例	135
6.1.1 实例分析: audacity	135
6.1.2 实例分析: fetchmail的-v选项	136
6.1.3 实例分析: gcc	139
6.1.4 实例分析: kmail	140
6.1.5 实例分析: sng	142
6.1.6 实例分析: terminfo数据库	144
6.1.7 实例分析: freeciv数据文件	146
6.2 为透明性和可显性而设计	148
6.2.1 透明性之禅	149
6.2.2 为透明性和可显性而编码	150
6.2.3 透明性和避免过度保护	151
6.2.4 透明性和可编辑的表现形式	152
6.2.5 透明性、故障诊断和故障恢复	153
6.3 为可维护性而设计	154
第7章 多道程序设计: 分离进程为独立的功能	157
7.1 从性能调整中分离复杂度控制	159
7.2 unix ipc 方法的分类	160
7.2.1 把任务转给专门程序	160
7.2.2 管道、重定向和过滤器	161
7.2.3 包装器	166
7.2.4 安全性包装器和bernie链	167
7.2.5 从进程	168
7.2.6 对等进程间通信	169
7.3 要避免的问题和方法	176
7.3.1 废弃的unix ipc方法	176
7.3.2 远程过程调用	178
7.3.3 线程——恐吓或威胁	180
7.4 在设计层次上的进程划分	181
第8章 微型语言: 寻找歌唱的乐符	183
8.1 理解语言分类法	185
8.2 应用微型语言	187
8.2.1 案例分析: sng	187
8.2.2 案例分析: 正则表达式	188
8.2.3 案例分析: glade	191
8.2.4 案例分析: m4	193
8.2.5 案例分析: xslt	194
8.2.6 案例分析: the documenter's workbench tools	195
8.2.7 案例分析: fetchmail的运行控制语法	199
8.2.8 案例分析: awk	200

8.2.9 案例分析: postscript	202
8.2.10 案例分析: bc和dc	203
8.2.11 案例分析: emacs lisp	205
8.2.12 案例分析: javascript	205
8.3 设计微型语言	206
8.3.1 选择正确的复杂度	207
8.3.2 扩展和嵌入语言	209
8.3.3 编写自定义语法	210
8.3.4 宏—慎用	210
8.3.5 语言还是应用协议	212
第9章 生成: 提升规格说明的层次	215
9.1 数据驱动编程	216
9.1.1 实例分析: ascii	217
9.1.2 实例分析: 统计学的垃圾邮件统计	218
9.1.3 实例分析: fetchmailconf中的元类改动	219
9.2 专用代码的生成	225
9.2.1 实例分析: 生成ascii显示的代码	225
9.2.2 实例分析: 为列表生成html代码	227
第10章 配置: 迈出正确的第一步	231
10.1 什么应是可配置的	231
10.2 配置在哪里	233
10.3 运行控制文件	234
10.3.1 实例分析: .netrc文件	236
10.3.2 到其它操作系统的可移植性	238
10.4 环境变量	238
10.4.1 系统环境变量	238
10.4.2 用户环境变量	240
10.4.3 何时使用环境变量	240
10.4.4 到其它操作系统的可移植性	242
10.5 命令行选项	242
10.5.1 从-a到-z的命令行选项	243
10.5.2 到其它操作系统的可移植性	248
10.6 如何挑选方法	248
10.6.1 实例分析: fetchmail	249
10.6.2 实例分析: xfree86服务器	251
10.7 论打破规则	252
第11章 接口: unix环境下的用户接口设计模式	253
11.1 最小立异原则的应用	254
11.2 unix接口设计的历史	256
11.3 接口设计评估	257
11.4 cli和可视接口之间的权衡	259
11.4.1 实例分析: 编写计算器程序的两种方式	262
11.5 透明度、表现力和可配置性	264
11.6 unix接口设计模式	266
11.6.1 过滤器模式	266
11.6.2 cantrip模式	268
11.6.3 源模式	268
11.6.4 接收器模式	269
11.6.5 编译器模式	269
11.6.6 ed模式	270
11.6.7 roguelike 模式	270
11.6.8 “引擎和接口分离” 模式	273
11.6.9 cli服务器模式	278
11.6.10 基于语言的接口模式	279

- 11.7 应用unix接口设计模式 280
- 11.7.1
- 11.8 网页浏览器作为通用前端 281
- 11.9 沉默是金 284
- 第12章 优化 287
  - 12.1 什么也别做，就站在那儿 287
  - 12.2 先估量，后优化 288
  - 12.3 非定域性之害 290
  - 12.4 吞吐量和延迟 291
    - 12.4.1 批操作 292
    - 12.4.2 重叠操作 293
    - 12.4.3 缓存操作结果 293
- 第13章 复杂度：尽可能简单，但别简过了头 295
  - 13.1 谈谈复杂度 296
    - 13.1.1 复杂度的三个来源 296
    - 13.1.2 接口复杂度和实现复杂度的折中 298
    - 13.1.3 必然的、可能的和偶然的复杂度 299
    - 13.1.4 映射复杂度 300
    - 13.1.5 当简洁性不能胜任 302
  - 13.2 五个编辑器的故事 302
    - 13.2.1 ed 304
    - 13.2.2 vi 305
    - 13.2.3 sam 306
    - 13.2.4 emacs 307
    - 13.2.5 wily 308
  - 13.3 编辑器的适当规模 309
    - 13.3.1 鉴别复杂度问题 309
    - 13.3.2 折衷无用 312
    - 13.3.3 emacs是个反unix传统的论据吗 314
  - 13.4 软件的适度规模 316
- part iii 319
- 第14章 语言：c还是非c 321
  - 14.1 unix下语言的丰饶 321
  - 14.2 为什么不是c 323
  - 14.3 解释型语言和混合策略 325
  - 14.4 语言评估 325
    - 14.4.1 c 326
    - 14.4.2 c++ 327
    - 14.4.3 shell 330
    - 14.4.4 perl 332
    - 14.4.5 tcl 334
    - 14.4.6 python 336
    - 14.4.7 java 339
    - 14.4.8 emacs lisp 342
  - 14.5 未来趋势 344
  - 14.6 选择x工具包 346
- 第15章 工具：开发的战术 349
  - 15.1 开发者友好的操作系统 349
  - 15.2 编辑器选择 350
    - 15.2.1 了解vi 351
    - 15.2.2 了解emacs 351
    - 15.2.3 非虔诚的选择：两者兼用 352
  - 15.3 专用代码生成器 352
  - 15.3.1 yacc和lex 353

15.3.2 实例分析：fetchmailrc的语法 356

15.3.3 实例分析：glade 356

15.4 make：自动化编译 357

15.4.1 make的基本理论 357

15.4.2 非c/c++开发中的make 359

15.4.3 通用生成目标 359

15.4.4 生成makefile 362

15.5 版本控制系统 364

15.5.1 为什么需要版本控制 364

15.5.2 手工版本控制 365

15.5.3 自动化的版本控制 366

15.5.4 unix的版本控制工具 367

15.6 运行期调试 369

15.7 性能分析 370

15.8 使用emacs整合工具 370

15.8.1 emacs和make 371

15.8.2 emacs和运行期调试 371

15.8.3 emacs和版本控制 371

15.8.4 emacs和profiling 372

15.8.5 像ide一样，但更强 373

第16章 重用：论不要重新发明轮子 375

16.1 猪小兵的故事 376

16.2 透明性是重用的关键 379

16.3 从重用到开源 380

16.4 生命中最美妙的就是“开放” 381

16.5 何处找 384

16.6 使用开源软件的问题 385

16.7 许可证问题 386

16.7.1 开放源码的资格 386

16.7.2 标准开放源码许可证 388

16.7.3 何时需要律师 390

part iv 391

第17章 可移植性：软件可移植性与遵循标准 393

17.1 c语言的演化 394

17.1.1 早期的c语言 395

17.1.2 c 语言标准 396

17.2 unix 标准 398

17.2.1 标准和unix之战 398

17.2.2 庆功宴上的幽灵 401

17.2.3 开源世界的unix标准 402

17.3 ietf和rfc标准化过程 403

17.4 规格dna，代码rna 405

17.5 可移植性编程 408

17.5.1 可移植性和编程语言选择 409

17.5.2 避免系统依赖性 412

17.5.3 移植工具 413

17.6 国际化 413

17.7 可移植性、开放标准以及开放源码 414

第18章 文档：向网络世界阐释代码 417

18.1 文档概念 418

18.2 unix风格 420

18.2.1 大文档偏爱 420

18.2.2 文化风格 421

18.3 各种unix文档格式 422

18.3.1 troff和documenter's workbench tools	422
18.3.2 tex	424
18.3.3 texinfo	425
18.3.4 pod	425
18.3.5 html	426
18.3.6 docbook	426
18.4 当前的混乱和可能的出路	426
18.5 docbook	427
18.5.1 文档类型定义	427
18.5.2 其它dtd	428
18.5.3 docbook 工具链	429
18.5.4 移植工具	431
18.5.5 编辑工具	432
18.5.6 相关标准和实践	433
18.5.7 sgml	433
18.5.8 xml-docbook 参考书籍	433
18.6 编写unix文档的最佳实践	434
第19章 开放源码：在unix新社区中编程	437
19.1 unix和开放源码	438
19.2 与开源开发者协同工作的最佳实践	440
19.2.1 良好的修补实践	440
19.2.2 良好的项目、档案文件命名实践	444
19.2.3 良好的开发实践	447
19.2.4 良好的发行制作实践	450
19.2.5 良好的交流实践	454
19.3 许可证的逻辑：如何挑选	456
19.4 为什么应使用某个标准许可证	457
19.5 各种开源许可证	457
19.5.1 mit或者x consortium许可证	457
19.5.2 经典bsd许可证	457
19.5.3 artistic许可证	458
19.5.4 通用公共许可证	458
19.5.5 mozilla 公共许可证	459
第20章 未来：危机与机遇	461
20.1 unix传统中的必然和偶然	461
20.2 plan 9：未来之路	464
20.3 unix设计中的问题	466
20.3.1 unix文件就是一大袋字节	466
20.3.2 unix对gui的支持孱弱	467
20.3.3 文件删除不可撤销	468
20.3.4 unix假定文件系统是静态的	469
20.3.5 作业控制设计拙劣	469
20.3.6 unix api 没有使用异常	470
20.3.7 ioctl(2)和fcntl(2)是个尴尬	471
20.3.8 unix安全模型可能太过原始	471
20.3.9 unix名字种类太多	472
20.3.10 文件系统可能有害论	472
20.3.11 朝向全局互联网地址空间	472
20.4 unix的环境问题	473
20.5 unix文化中的问题	475
20.6 信任的理由	477
附录a 缩写词表	479
附录b 参考文献	483
附录c 贡献者	495

附录d 无根的根：无名师的unix心传 499

colophon 510

索引 511

· · · · · (收起)

[UNIX编程艺术 下载链接1](#)

标签

UNIX

编程

哲学

计算机

程序设计

经典

计算机科学

Linux

评论

地铁上粗读，这些好听得耳朵都磨出茧子了，其实我更想读 Unix 的设计缺陷=。 =

---

这本书关于模式、规则的理解，达到了一个较高的境界。

-----  
这个是新翻译的版本 还是只重新印刷的？见鬼

-----  
: TP316.81/1162-1

-----  
应当一年一读。

-----  
早读早好

-----  
读了一半。感觉实在是太老了，有点过时

-----  
当成一部历史书来读比较好。了解Unix的文化，能用上就用，不能用上就了解一下，扩展知识面非常不错。

-----  
说的是Unix的文化与哲学

-----  
很多东西目前还无法理解，工作几年之后会重读的。

-----  
部分翻译太屎了

-----  
南图 MS没有留下什么东西，需再读

-----  
喜欢这本书的前半部分，后面的部分很多软件没有实际用过，感悟不深。

-----  
他不是技术手册，他是哲学教材

-----  
这本书很好都说明了什么才是Unix风格，什么才是正宗的Unix程序员。

-----  
大牛作品,值得一读...说是编程,实际基本没代码在里面,主要介绍编程哲学,不过很受用.

-----  
对win系统及尽揶揄之能事 过瘾

-----  
在地铁、公交车、床上读完的

-----  
学习UNIX的哲学

-----  
没啥收获，啰里啰嗦

-----  
[UNIX编程艺术 下载链接1](#)

## 书评

贯穿始终的 KISS

原则，很多年前就被谆谆教导过了。它被我无时无刻的都拿出来警告自己的设计过程。  
读完这本书，让我对 KISS

又有了一次升华。其实，这本书对我几个月来设计游戏服务器架构的影响是满大的。坚定了我每写一个程序做好一件事的决心。让我更确信用多进程的设计取代...

作为一个多年的开源 (Open Source) 拥趸，像《UNIX编程艺术》 (The Art of Unix Programming) 这样的好书自然不能错过。大约一周前我无意中在公司书柜中发现了它，立刻开始投入阅读。现在，我已经开始边读第二遍边作读书笔记了。  
开篇的译序很有趣，第一句话就写道：“...

---

This book reveals the history, art, culture, philosophy, practices, guideline about programming (with) Unix, from the OS itself to the programming languages. It is not doubt that Unix is one of the most influencing OS ever built. Just as mentioned in the...

---

所以在这里可以读到正版的：<http://www.faqs.org/docs/artu/>  
当然如果不是非要在网上读这样需要思考一下的书不可，买一本还是很值得的。  
这本书主要偏向软件工程的角度。在joelonsoftware上有一篇很详细的review（我也是看那篇review才想要读这本书）。其中提到Windows（或者...）

---

本文同步发于我的blog: <http://www.vingel.com>  
这本书我已经买了三个星期，一直在看。以前看它的英文版，仅仅走马观花般看了一遍。  
。现在这个中文版，目前为止我已经看了三遍，而每次都会有新的体会。我得到的关于《Unix编程艺术》最深的感想就是：Unix中无所不在的K.I.S.S(Keep...

---

书籍拿到手比较厚实，感觉很有分量，不过读起来倒不那么难啃。  
翻译的还是很流畅，整体下来很酣畅，译者说用了1年，看来还是比较恰当的。  
书中介绍了unix的文化、历史，举了不少案例。  
ESR身上unix黑客气味贯彻始终本书，在ESR看来unix编程就是个玩的过程。  
本书中融贯了n...

---

买这本书主要冲两点，第一是封面上的Software Development Productivity Award标志，第二是作者Eric Raymond，Raymond是开源运动的发起者之一，他的经典文章“大教堂和市集”广为流传。由于没有急用而且是英文，书买来后看了目录就收到书架上。  
最近又有项目要用linux...

这本书买的比较晚，却是赶在其他书之前最先读完。前前后后估计读了三个月有余，当然因为中途有好看的小说插队，以至于耽搁了不少。

这个标题很容易让人以为和那部旷世巨作一样，还好通过评论，已经知道是一种思想的阐述。所以刚才，当我把书放回书架的时候，犹豫了一下，还是...

这本书不是技术类书籍，不是Knuth的The Art of Computer

Programming的姊妹篇，而是一本关于黑客文化的书。所以，适合对这些技术有些了解或感兴趣的人，睡前拿来翻一翻。书里没有数学公式，甚至连代码也没有。有的是大段大段的历史和引文，成与败，得与失的比较，最终落到开篇...

这本书是一种智者的言论，作者的经验学识让他所见所想都不同于我们，当一切成为历史，留下的沉淀下来的是这些闪光的思想。

但对于性能应该放到最后才进行考虑，有见仁见智的观点，对于性能应该视之为等同于风险来评估预测和管理控制，特别是对于大型软件，开发周期长，人员多，...

500来页的书以五个晚上的时间飙完，本身就说明了其简单。

是最近以来看得最爽利的两本软件书之一（另一本是云风的《我的编程感悟》），全书是对“主流”软件工程的反动，但每每契合吾意，往往有醍醐灌顶之感。  
ESR属于优点和缺点都很明显的作者，还好这次有心理...

如果早三年就去读，我的人生必将不同。如果早五年来读，大概我还读不懂。

写程序做项目，许多事情没有经历过，是很难体会的。（刚写了一大段攻击C++的文字，怕引起圣战，就删了。此处省略300字。）

好吧，总之，不论经验丰富还是初出茅庐，都建议读一读。也建议过一两年再捡...

3.1.3 Cooperating Processes it would not have been trivial without the fundamental unifying notion of the process as an autonomous unit of computation, with process control being programmable.

这句被翻译成了：……进程是自主运算单元的统一性记号…… 你能...

<https://gcd0318.wordpress.com/2016/04/20/%E5%8F%88%E5%88%B7%E4%BA%86%E4%B8%80%E9%81%8Dtaoup/> 2016年04月20日 又刷了一遍taoup Filed under: 感悟 — gcd0318 @ 04:10

十年以来每隔两三年就把这本书拿出来再读一遍，而且一遍比一遍读的快，一方面是我自己的领悟提升了，...

目前读到了第13章，中文版的。

如果想了解Unix的哲学，并且从多个视角去了解Unix，无疑，TAUP是一个很好的起点。不管是Unix的设计原则，还是Unix的诞生历史，ESR都信手拈来。每个章节的组织都是松耦合的，这也给了读者极大的阅读自由！

从作者写书到我读这本书已经事隔十年。

对于一个使用linux有2年的人（其实我一直不认为经验年限和对一件东西的了解程度有线性关系），我阅读完之后的感受是：有些观念很深入我心，有一些却不了解，或者根本提不起兴趣看。还是说说那些深入我心的感悟。1，机制和策略的分离，...

<http://herpolhode.com/rob/ugly.pdf> The Good, the Bad, and the Ugly: The Unix! Legacy high-level programming language hierarchical file system uniform, unformatted files (text) separable shell distinct tools pipes regular expressions portability security ...

原著成书于2003年，而且作者说写了5年。从2003到现在2011已经又8年了，许多事情又变化了。。。

这本书在学校就读过，但当时也就是读过一遍，略微记住几个名词而已，比如KISS和文本化，现在回想下，这两个概念对自己影响确实蛮深的，当初没有白读。最初阅读时，好多软件都没有接...

内容涵盖从philosophy 到 practical issues。其实K.I.S.S.

的概念很简单，容易理解，但是就知道K.I.S.S. 的概念是远不够的，理解并运用K.I.S.S. 才是我们的最终目标。本书包含的很多Case study 非常值得一看，里面包含了多年实践的经验，已经对未来软件设计的指导，very nice.

[UNIX编程艺术 下载链接1](#)