

<<UNIX编程艺术>>

图书基本信息

书名：<<UNIX编程艺术>>

13位ISBN编号：9787121176654

10位ISBN编号：7121176653

出版时间：2012-8

出版时间：电子工业出版社

作者：埃瑞克·S.理曼德

页数：530

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

前言

出版说明 悦读上品得乎益友 孔子云：“取乎其上，得乎其中；取乎其中，得乎其下；取乎其下，则无所得矣”。

对于读书求知而言，这句古训教我们去读好书，最好是好书中的上品——经典书。其中，科技人员要读的技术书，因为直接关乎客观是非与生产效率，阅读选材本更应慎重。然而，随着技术图书品种的日益丰富，发现经典书越来越难，尤其对于涉世尚浅的新读者，更为不易，而他们又往往是最需要阅读、提升的重要群体。

所谓经典书，或说上品，是指选材精良、内容精练、讲述生动、外延丰盈、表现手法体贴入微的读品，它们会成为读者的知识和经验库中的重要组成部分，并且拥有从不断重读中汲取养分的空间。因此，选择阅读上品的问题便成了有效阅读的首要问题。当然，这不只是效率问题，上品促成的既是对某一种技术、思想的真正理解和掌握，同时又是一种感悟或享受，是一种愉悦。

与技术本身类似，经典IT技术书多来自国外。深厚的积累、良好的写作氛围，使一批大师为全球技术学习者留下了璀璨的智慧瑰宝。就在那个年代即将远去之时，无须回眸，也能感受到这一部部厚重而深邃的经典著作，在造福无数读者后从未蒙尘的熠熠光辉。

而这些凝结众多当今国内技术中坚美妙记忆与绝佳体验的技术图书，虽然尚在国外图书市场上大放异彩，却已逐渐淡出国人的视线。

最为遗憾的是，迟迟未有可以填补空缺的新书问世。

而无可替代，不正是经典书被奉为圭臬的原因？

为了不让国内读者，尤其是即将步入技术生涯的新一代读者，就此错失这些滋养过先行者们的

好书，以出版IT精品图书，满足技术人群需求为己任的我们，愿意承担这一使命。

本次机遇惠顾了我们，让我们有机会携手权威的Pearson公司，精心推出“传世经典书丛”。

在我们眼中，“传世经典”的价值首先在于——既适合喜爱科技图书的读者，也符合专家们挑剔的标准。

幸运的是，我们的确找到了这些堪称上品的佳作。

丛书带给我们的幸运颇多，细数一下吧。

得以引荐大师著作 有恐思虑不周，我们大量参考了国外权威机构和网站的评选结果，并得到了Pearson的专业支持，又进一步对符合标准之图书的国内外口碑与销售情况进行细致分析，也听取了国内技术专家的宝贵建议，才有幸选出对国内读者最富有技术养分的大师上品。

向深邃的技术内涵致敬 中外技术环境存在差异，很多享誉国外的好书未必适用于国内读者；且技术与应用瞬息万变，很容易让人心生迷惘或疲于奔命。

本丛书的图书遴选，注重打好思考方法与技术理念的根基，旨在帮助读者修炼内功，提升境界，将技术真正融入个人知识体系，从而可以一通百通，从容面对随时涌现的技术变化。

翻译与评注的双项选择 引进优秀外版著作，将其翻译为中文供国内读者阅读，较为有效与常见。

但另有一些外语水平较高、喜好阅读原版的读者，苦于对技术理解不足，不能充分体会原文表述的

精妙，需要有人指导与点拨。

而一批本土技术精英经过长期经典熏陶及实践锤炼，已足以胜任这一工作。有鉴于此，本丛书在翻译版的同时推出融合英文原著与中文点评、注释的评注版，供不同志趣的读者自由选择。

承蒙国内一流译（注）者的扶持 优秀的英文原著最终转化为真正的上品，尚需跨越翻译鸿沟，外版图书的翻译质量一直屡遭国内读者诟病。

评注版的增值与含金量，同样依赖于评注者的高卓才具。

好在，本丛书得到了久经考验的权威译（注）者的认可和支持，首肯我们选用其佳作，或亲自参与评注工作。

正是他们的参与保证了经典的品质，既再次为我们的选材把关，更提供了一流的中文表述。

期望带给读者良好的阅读体验 一本好书带给人的愉悦不止于知识收获，良好的阅读感受同样不可缺少，且对学业不无助益。

为了让读者收获与上品相称的体验，我们在图书装帧设计与选材用料上同样不敢轻率，惟愿送到读者手中的除了珠玑章句，还有舒适与熨帖的视觉感受。

所有参与丛书出版的人员，尽管能力有限，却无不心怀严谨之心与完美愿望。

如果读者朋友能从潜心阅读这些上品中偶有获益，不啻为对我们工作的最佳褒奖。

若有阅读感悟，敬请拨冗告知，以鼓励我们继续在这一道路上贡献绵薄之力。

如有不周之处，也请不吝指教。

电子工业出版社博文视点 二〇一〇年十二月

<<UNIX编程艺术>>

内容概要

《传世经典书丛：UNIX编程艺术》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。

<<UNIX编程艺术>>

作者简介

作者:(美)Raymond

<<UNIX编程艺术>>

书籍目录

| | |
|---|-----|
| 序 Part I | 1 |
| 第1章 哲学 | 3 |
| 1.1 文化？ | 3 |
| 1.2 Unix的生命力 | 4 |
| 1.3 反对学习Unix文化的理由 | 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 UnixHack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 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：news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 RFC822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 WindowsINI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 UnixIPC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Bernstein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UnixIPC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：TheDocumenter's work bench 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：EmacsLisp205 8.2.12 案例分析：JavaScript205 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：ascii217 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：fetchmail249 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 多价程序模式 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 ed304 13.2.2 vi305 13.2.3 Sam306 13.2.4 Emacs307 13.2.5 Wily308 13.3 编辑器的适当规模309 13.3.1 甄别复杂度问题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 319 第14章 语言：C还是非C321 14.1 Unix下语言的丰饶321 14.2 为什么不是C323 14.3 解释型语言和混合策略325 14.4 语言评估325 14.4.1 C326 14.4.2 C++327 14.4.3 Shell330 14.4.4 Perl332 14.4.5 Tcl334 14.4.6 Python336 14.4.7 Java339 14.4.8 EmacsLisp342 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vi351 15.2.2 了解Emacs351 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lex353 15.3.2 实例分析：fetchmailrc的语法356 15.3.3 实例分析：Glade356 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make359 15.4.3 通用生成目标359 15.4.4 生成Makefile362 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make371 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Profiling372 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 生命中最美好的就是“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 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RNA405 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 Work bench Tools422 18.3.2 TEX424 18.3.3 Texinfo425 18.3.4 POD425 18.3.5 HTML426 18.3.6 DocBook426 18.4 当前的混乱和可能的出路426 18.5 DocBook427 18.5.1 文档类型定义427 18.5.2 其它DTD428 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 SGML433 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Xconsortium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 Plang：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 UnixAPI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 Colophon510 索引511

章节摘录

版权页：插图：terminfo本身使用文件系统作为一个简单的层级数据库。

这种偷懒相当具有建设性，符合经济性原则和透明性原则。

这意味着对文件系统进行浏览、检查和修改的所有普通工具都可以用于对terminfo数据库进行浏览、检查和修改；无需编写和调试专用工具（用于打包和解包单个记录的tic（1）和infocmp（1）工具除外）。

这也意味着要加速数据库的访问就得要加速文件系统本身，知道这一点可以使更多应用程序受益，而不仅仅是curses（3）的用户。

这种结构还有另外一种优点，但在terminfo例子中没有展示出来：你开始使用Unix的授权机制而不用自己编写带来额外负担的访问控制层。

这也是采纳而不是对抗Unix“一切皆文件”基本原则的结果。

terminfo目录的布局在大多数Unix文件系统上都很浪费空间。

每条目长度通常在400~1400字节之间，但是文件系统通常为每一个非空磁盘文件至少分配4k的空间。

出于选择压缩二进制格式的同一个原因，即为了把terminfo使用的程序的启动延时降到最小，设计者接受了这个代价。

同一价格所能买到的磁盘容量已经猛增了一千倍，更能证明这个决定的正确。

比较这种格式和Microsoft Windows的注册表文件所用的格式很有启发意义。

注册表是Windows本身及应用程序都使用的属性数据库。

所有注册记录都存放在一个大文件中。

注册记录既包含文本也包含二进制数据，需要专用的编辑工具。

别的不说，这种“一个大文件”的方法还导致了臭名昭著的“注册表蠕变”现象；平均访问时间随着新记录的加入而无限上升。

因为系统没有提供标准API来编辑注册表，应用程序本身使用专用代码编辑注册表，使得注册表极易受损，甚至能够锁定整个系统。

使用Unix文件系统作为数据库是一种策略，对数据库要求简单的其它应用程序可以效仿并从中受益。

不这样做的充分理由通常与性能问题无关，更可能的情形是数据库关键字不太适合做文件名。

无论如何，这是在原型设计时非常有用的一种很好的快速编程方法。

6.1.7 实例分析：Freeciv数据文件 Freeciv是一款受到Sid Meier经典的Civilization H启发而制作的开源策略游戏。

在该游戏中，每个玩家从一群到处流浪的新石器游牧民开始缔造一个文明。

玩家的文明可以探索并拓殖世界，参与战争，从事贸易和研究先进技术。

有些玩家实际上可能是人工智能；和这些电脑玩家玩单机游戏很有挑战性。

如果谁统治了整个世界，或者第一个研制出先进技术从而获得宇宙飞船飞往半人马座阿尔法星（Alpha Centauri），谁就是游戏的胜利者。

源码和文档可以在处获得。

<<UNIX编程艺术>>

编辑推荐

《传世经典书丛:UNIX编程艺术》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。

<<UNIX编程艺术>>

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>