

The Art of UNIX Programming

Eric S. Raymond

With contributions
from UNIX legends
Thompson, Kernighan,
McIlroy, Arnold,
Bellovin, Korn,
Gettys, Packard,
Lesk, Feldman,
McKusick,
and Spencer



ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

The Art of Unix Programming

Art of UNIX Programming, The, Portable Documents

Table of Contents

Contents

Preface

I: Context

1 Philosophy: Philosophy Matters

- 1.1 Culture? What Culture?
- 1.2 The Durability of Unix
- 1.3 The Case against Learning Unix Culture
- 1.4 What Unix Gets Wrong
- 1.5 What Unix Gets Right
- 1.6 Basics of the Unix Philosophy
- 1.7 The Unix Philosophy in One Lesson
- 1.8 Applying the Unix Philosophy
- 1.9 Attitude Matters Too

2 History: A Tale of Two Cultures

- 2.1 Origins and History of Unix, 1969-1995
- 2.2 Origins and History of the Hackers, 1961-1995
- 2.3 The Open-Source Movement: 1998 and Onward
- 2.4 The Lessons of Unix History

3 Contrasts: Comparing the Unix Philosophy with Others

- 3.1 The Elements of Operating-System Style
- 3.2 Operating-System Comparisons
- 3.3 What Goes Around, Comes Around

Table of Contents

II: Design

4 Modularity: Keeping It Clean, Keeping It Simple

- 4.1 Encapsulation and Optimal Module Size
- 4.2 Compactness and Orthogonality
- 4.3 Software Is a Many-Layered Thing
- 4.4 Libraries
- 4.5 Unix and Object-Oriented Languages
- 4.6 Coding for Modularity

5 Textuality: Good Protocols Make Good Practice

- 5.1 The Importance of Being Textual
- 5.2 Data File Metaformats
- 5.3 Application Protocol Design
- 5.4 Application Protocol Metaformats

6 Transparency: Let There Be Light

- 6.1 Studying Cases
- 6.2 Designing for Transparency and Discoverability
- 6.3 Designing for Maintainability

7 Multiprogramming: Separating Processes to Separate Function

- 7.1 Separating Complexity Control from Performance Tuning
- 7.2 Taxonomy of Unix IPC Methods
- 7.3 Problems and Methods to Avoid
- 7.4 Process Partitioning at the Design Level

8 Minilanguages: Finding a Notation That Sings

- 8.1 Understanding the Taxonomy of Languages
- 8.2 Applying Minilanguages
- 8.3 Designing Minilanguages

9 Generation: Pushing the Specification Level Upwards

Table of Contents

- 9.1 Data-Driven Programming
- 9.2 Ad-hoc Code Generation
- 10 Configuration: Starting on the Right Foot
 - 10.1 What Should Be Configurable?
 - 10.2 Where Configurations Live
 - 10.3 Run-Control Files
 - 10.4 Environment Variables
 - 10.5 Command-Line Options
 - 10.6 How to Choose among the Methods
 - 10.7 On Breaking These Rules
- 11 Interfaces: User-Interface Design Patterns in the Unix Environment
 - 11.1 Applying the Rule of Least Surprise
 - 11.2 History of Interface Design on Unix
 - 11.3 Evaluating Interface Designs
 - 11.4 Tradeoffs between CLI and Visual Interfaces
 - 11.5 Transparency, Expressiveness, and Configurability
 - 11.6 Unix Interface Design Patterns
 - 11.7 Applying Unix Interface-Design Patterns
 - 11.8 The Web Browser as a Universal Front End
 - 11.9 Silence Is Golden
- 12 Optimization:
 - 12.1 Dont Just Do Something, Stand There!
 - 12.2 Measure before Optimizing
 - 12.3 Nonlocality Considered Harmful
 - 12.4 Throughput vs. Latency
- 13 Complexity: As Simple As Possible, but No Simpler
 - 13.1 Speaking of Complexity
 - 13.2 A Tale of Five Editors

Table of Contents

13.3 The Right Size for an Editor

13.4 The Right Size of Software

III: Implementation

14 Languages: To C or Not To C?

14.1 Unixs Cornucopia of Languages

14.2 Why Not C?

14.3 Interpreted Languages and Mixed Strategies

14.4 Language Evaluations

14.5 Trends for the Future

14.6 Choosing an X Toolkit

15 Tools: The Tactics of Development

15.1 A Developer-Friendly Operating System

15.2 Choosing an Editor

15.3 Special-Purpose Code Generators

15.4 make: Automating Your Recipes

15.5 Version-Control Systems

15.6 Runtime Debugging

15.7 Profiling

15.8 Combining Tools with Emacs

16 Reuse: On Not Reinventing the Wheel

16.1 The Tale of J. Random Newbie

16.2 Transparency as the Key to Reuse

16.3 From Reuse to Open Source

16.4 The Best Things in Life Are Open

16.5 Where to Look?

16.6 Issues in Using Open-Source Software

16.7 Licensing Issues

IV: Community

Table of Contents

17 Portability: Software Portability and Keeping Up Standards

- 17.1 Evolution of C
- 17.2 Unix Standards
- 17.3 IETF and the RFC Standards Process
- 17.4 Specifications as DNA, Code as RNA
- 17.5 Programming for Portability
- 17.6 Internationalization
- 17.7 Portability, Open Standards, and Open Source

18 Documentation: Explaining Your Code to a Web-Centric World

- 18.1 Documentation Concepts
- 18.2 The Unix Style
- 18.3 The Zoo of Unix Documentation Formats
- 18.4 The Present Chaos and a Possible Way Out
- 18.5 DocBook
- 18.6 Best Practices for Writing Unix Documentation

19 Open Source: Programming in the New Unix Community

- 19.1 Unix and Open Source
- 19.2 Best Practices for Working with Open-Source Developers
- 19.3 The Logic of Licenses: How to Pick One
- 19.4 Why You Should Use a Standard License
- 19.5 Varieties of Open-Source Licensing

20 Futures: Dangers and Opportunities

- 20.1 Essence and Accident in Unix Tradition
- 20.2 Plan 9: The Way the Future Was
- 20.3 Problems in the Design of Unix
- 20.4 Problems in the Environment of Unix
- 20.5 Problems in the Culture of Unix

Table of Contents

20.6 Reasons to Believe

A: Glossary of Abbreviations

B: References

C: Contributors

D: Rootless Root: The Unix Koans of Master Foo

Colophon

Index