The Art of Unix Programming Teaching Guide

by Eric Steven Raymond

by Rob Landley

The Art of Unix Programming Teaching Guide

by Eric Steven Raymond
by Rob Landley

Table of Contents

How to use this guide

The Art Of Unix Programming (TAOUP) was written primarily to be used as a self-education tool for professional or enthusiatic amateur programmers. It was not designed to be a course textbook, but it may usefully be employed as one. This instructional guide is intended to help.

TAOUP is different from the general run of Unix tutorials, which tend to concentrate on the level of basic shell usage, application programming interfaces and other relatively low-level issues. TAOUP very deliberately focuses at a higher level: design principles, style, and the history that has informed both. It is not a replacement for an introductory UNIX textbook, but a valuable complement to same.

TAOUP could be appropriate in at least three different contexts:

- *Courses on the Unix operating system.* TAOUP is appropriate as a core text in a course on Unix, especially one oriented toward programmers (as opposed to, say, training system administrators).
- *Intermediate-to-advanced courses on software engineering.* The problems of complexity control in larger software systems, and the lesson Unix experience teaches about that control, are a core theme of the book.
- *Courses on the history of computing.* The material is presented in a historical frame, with an emphasis on technology as a human activity. The evolution of Unix makes an interesting lens through which to examine interactions between technology, economics, and subcultural history.

The guide's recommendations are divided by chapters. For each chapter, we'll supply suggested exercises, concepts and definitions, short-form questions, and essay questions. We have organized the material using Benjamin Bloom's hierarchy of functions:

Knowledge	Observation and recall of information; knowledge of dates, events, places; knowledge of major ideas; mastery of facts.
Comprehension	Understanding information; grasp meaning; translate knowledge into new context; interpret facts, compare, contrast; order, group, infer causes; predict consequences.

Application	Use information; use methods, concepts, theories in new situations; solve problems using required skills or knowledge.
Analysis	Seeing patterns; organization of parts; recognition of hidden meanings; identification of components.
Synthesis	Use old ideas to create new ones; generalize from given facts; relate knowledge from several areas; predict, draw conclusions.
Evaluation	Compare and discriminate between ideas; assess value of theories, presentations, make choices based on reasoned argument, verify value of evidence, recognize subjectivity.

For the exercises and Analysis/Synthesis/Evaluation questions it is assumed that students will have access to a Web browser while working on them.

Please help us improve this teaching guide. If it inspires you to develop other questions, exercises, and materials relating to this book, send us a copy at `<esr+taoupguide@thyrsus.com>`. Contributors will be credited in future editions of the guide.

Chapter 1. Philosophy

Outline

Chapter 1, Philosophy:

- Culture? What Culture?
    - Unix has a strong technical culture.
        - This is like a mature engineering discipline.
        - Most kinds of programming are *not* like mature engineering disciplines.
- The Durability of Unix
    - Unix has survived for 30 years, adapted to a wider range of different uses than any other operating system. It is in use on literally hundreds of hardware platforms, from mainframes to kitchen appliances. Over the years, the successful design elements of Unix have been tested and refined, and new ideas incorporated, while many passing fads have come and gone.
    - The C programming language was invented alongside Unix, and initially spread with it.
- The Case Against Learning Unix Culture
    - After decades of civil war, proprietary Unix is generally about as interesting as COBOL.

    - Open-source Unix has faced an uphill battle to attain desktop market share, and is at mid-2003 still slightly behind the Macintosh.
        - MacOS X is a promising but proprietary Unix
        - The high-end “workstation” market is perpetually under assault by commodity PC hardware.
    - Open-source Unix is developing rapidly; the case against it is not a very strong one.
- What Unix Gets Wrong
    - Technical bits covered later.
    - “Mechanism not policy” can be taken too far, delegating policy decisions to non-technical end users. (No policy can become a policy.)
        - Intelligent defaults aren’t a bad thing.
        - Linux distributors often address this.
- What Unix Gets Right
    - Open-Source Software
        - This is where the action is today.

- Cross-Platform Portability and Open Standards
    - Not just buzzwords, it actually works here. That’s *why* it’s lasted 30 years.
- The Internet and the World Wide Web
    - Unix has run the internet since 1980.
    - TCP/IP, web browsers, POP and IMAP email...
- The Open-Source Community
    - Taking “code reuse” to its logical conclusion.
- Flexibility All the Way Down
    - It’s been used to do everything. Unix is a jack of all trades, and has learned not to get too attached to any one special-case niche.
- Unix is Fun to Hack.
    - The open-source hobbyists could have reverse enginneered any OS. They uniformly chose Unix.

    - The Lessons of Unix Can Be Applied Elsewhere
        - Unix encourages and rewards good programming habits.
- Basics of the Unix Philosophy
    - Advice from the experts
        - Do one thing and do it well.
        - Expect output of programs to become input of other programs.
            - Text streams are a universal interface.
        - You can’t properly evaluate your design before you have a prototype running.
            - Premature optimization is the root of all evil. Profile, *then* optimize.
        - Ken Thompson said “When in doubt, use brute force.”
            - Fancy algorithms hide bugs.
            - The usual case is small data sets.

  - Building tools is almost never wasted effort.
  - Data sets, not algorithms, are central to programming.

- Rules

  - Rule of Modularity: Write simple parts connected by clean interfaces.
  - Rule of Clarity: Clarity is better than cleverness.
  - Rule of Composition: Design programs to be connected with other programs.
  - Rule of Separation: Separate policy from mechanism; separate interfaces from engines.
  - Rule of Simplicity: Design for simplicity; add complexity only where you must.
  - Rule of Parsimony: Write a big program only when it is clear by demonstration that nothing else will do.
  - Rule of Transparency: Design for visibility to make inspection and debugging easier.
  - Rule of Robustness: Robustness is the child of transparency and simplicity.
  - Rule of Representation: Fold knowledge into data, so program logic can be stupid and robust.
  - Rule of Least Surprise: In interface design, always do the least surprising thing.
  - Rule of Silence: When a program has nothing surprising to say, it should say nothing.
  - Rule of Repair: Repair what you can - but when you must fail, fail noisily and as soon as possible.
  - Rule of Economy: Programmer time is expensive; conserve it in preference to machine time.

    - Rule of Generation: Avoid hand-hacking; write programs to write programs when you can.
    - Rule of Optimization: Prototype before polishing. Get it working before you optimize it.
    - Rule of Diversity: Distrust all claims for one true way.
    - Rule of Extensibility: Design for the future, because it will be here sooner than you think.

- The Unix Philosophy in One Lesson
  - K.I.S.S.: Keep It Simple, Stupid
- Applying the Unix philosophy
  - There is a list of bullet points on the page.
- Attitude Matters Too
  - The best Unix hackers are as driven as Olympic athletes. It's not their day job, it's what their day job gives them time to do.

Lecture themes

- Unix has had an exceptionally long lifetime, and consequently has a richer surrounding culture than most other software technologies.
- Unix is central to the technology of the Internet and the World Wide Web.

- The strengths of Unix are grounded in elegance, playfulness, and pragmatism.
- The design rules introduce themes that will recur throughout the book. It may be helpful to group them as follows:

  Modularity, Clarity, Composition, Separation, Simplicity, Parsimony, Transparency, Robustness, Representation, These are about design, with the unifying theme of keeping things simple.

  Least Surprise, Silence. These are about user and developer interfaces.
  D1Repair, Economy, Generation, Optimization. The focus in these is more on coding practice.

  Diversity, Extensibility. Linking theme: systems have to be open, flexible, and capable of evolving — otherwise they die.
- “Keep It Simple, Stupid” is an important maxim in other kinds of engineering as well. Applications can be found everywhere from big civil-engineering efforts to everyday consumer products.

Suggested Exercises

Have students match the names of the Unix design rules with their contents.

Concepts and definitions

mechanism vs. policy	Mechanism is how you do things. Policy is what you choose to do.
POSIX	The IEEE Portable Operating System Standard.
KISS	Keep It Simple, Stupid!

Knowledge questions

1. Name the common language of systems programming in which Unix is written.

C.

2. Supporting an extremely general set of graphics operations and deferring decisions about toolkits and interface look-and-feel is an instance of what Unix principle?

   Separate mechanism from policy.

3. Approximately how long has Unix been in use?

   As of 2003, a bit more than three decades.

4. Name a variant of Unix other than Linux.

   BSD and Solaris are mentioned in the chapter. Other acceptable answers would include AIX, XENIX, HP-UX, or Ultrix.

5. Did the Unix and Internet cultures begin to merge before or after the arrival of mass-market Internet service providers?

   Before, around 1980.

6. What is Ken Thompson’s way of summarizing advice to use simple algorithms unless there is a specific reason for complex ones?

   “When in doubt, use brute force.”

7. What is Rob Pike’s Rule Six?

   “There is no Rule Six.”

8. What does POSIX stand for?

   Portable Operating System Standard.

Essay questions

1. (Comprehension) What subject do the Rule of Composition, the Rule of Least Surprise and the Rule of Silence have in common that distinguishes them from the other rules?

2. (Comprehension) Several of the Unix design rules are attempts to avoid circumstances in which someone trying to understand and maintain a program will never have to digest so much at once that serious mental fatigue sets in. List three of the rules for which this is an important theme. Discuss how they reinforce each other.

3. (Comprehension) Explain the kind of user-interface problems that Unix's "mechanism, not policy" approach tends to produce.

4. (Application) What does the Rule of Diversity suggest about the Java programming environment?

5. (Application) What does the Rule of Silence suggest about the design of aircraft control panels? Why might violating it be lethally dangerous?

6. (Application) Apply the Rule of Modularity to the design of an automobile. Why does it seem unimportant with respect to a cheap, throwaway item like (say) a ballpoint pen?

7. (Analysis) The Rule of Generation and the Rule of Optimization could be viewed as special cases of the Rule of Economy. Pick either and explain why it is valuable in terms *other* than conserving programmer time.

8. (Analysis) Explain how the Rule of Simplicity differs from the Rule of Clarity. Under what circumstances might a program be clear to a programmer reading it, but not as simple as possible? Under what circumstances might a program be simple, but not clear?

9. (Synthesis) Software is more malleable than matter; it takes much more energy to rearrange bulk quantities of atoms than it does to change the state of bits in a computer. Suppose this were to change, and we could write and modify configurations of atoms nearly as easily as we do computer

programs. In this future, would the Rule of Separation apply to designing furniture? If not, why not?

10. (Synthesis) Generalize the prescription “Whenever possible, prototype in an interpreted language before coding C.” by identifying an appropriate category of languages, of which C is a member.

11. (Evaluation) Given a choice between (a) a program that can be closely tailored to your needs but is complicated to configure, and (b) a simple program with a “one size fits all” that does almost but not quite what you want, which would you choose? Briefly explain your choice. Now imagine that you had made the opposite choice and briefly explain that one. For each case, feel free to pick a particular kind of application software that you actually use.

12. (Evaluation) What are the characteristics of a mature engineering discipline? Pick two engineering specialties, one well-established and mature (like civil, chemical, or mechanical engineering), one in an early growth phase (like genomics, or wireless digital networking). Compare.

Chapter 2. History

Outline

Suggested Exercise

Have students match these figures to their roles as described in the chapter:

Ken Thompson	The architect of Unix.
Dennis Ritchie	Architect of C, co-inventor of Unix.
Bill Jolitz	Developer of 386BSD.
Keith Bostic	Cleaned AT⟼T out of the BSD sources.
Bill Joy	Key Berkeley developer, co-founder of Sun Microsystems.
John McCarthy	Invented the Lisp language and the concept of timesharing.
Ken Olsen	CEO of DEC, described Unix as “snake oil”.
Eric Raymond	Theorist/evangelist of open source and the ‘bazaar’ model.
Richard M. Stallman	Visionary, moralist, author of the GPL.
Linus Torvalds	Inventor of Linux.
Larry Wall	Inventor of patch(1).

Concepts and definitions

Bell Laboratories	The research arm of the Bell Telephone monopoly; for many years one of the world's most effective and prestigious industrial-research laboratories. The transistor was invented there in 1948, and the laser around 1960.
Multics	The immediate ancestor of Unix. A first attempt at creating an 'information utility'.
ARPANET	The first wide-area network; ancestor of the Internet.
Version 7	Latest common ancestor of all modern Unixes.
TCP/IP	Base-level protocol of the Internet.
GNU project	An effort to create an entire Unix-workalike operating system under non-proprietary licensing.
open source	A development style in which all source code is made publicly available for peer review.

Knowledge questions

1. Who were the two co-inventors of Unix

   Ken Thompson and Dennis Ritchie

2. What important networking technology was born the same year as Unix in 1969?

   The ARPANET, the ancestor of today's Internet.

3. What did ‘UNICS’ stand for?

   UNiplexed Information and Computing Service.

4. In what year was Unix rewritten in C?

   1973

5. Name the university that put TCP/IP on Unix after 1980.

   The University of California at Berkeley.

6. In what year did MIT take delivery of the first PDP-1?

   1961.

7. In what year was the Free Software Foundation founded?

   1985.

8. Name two pioneering open-source development projects hosted at MIT.

   The GNU project and the X windowing system.

Essay questions

1. (Comprehension) The general public became aware of the Internet around 1994-1995, after the invention of the World Wide Web in 1991 and the first wave of local Internet Service Providers launched in 1992-1993, largely in basements and garages. To what extent were inexpensive but powerful machines able to do networking a prerequisite for all three developmenrs? Discuss.

2. (Comprehension) The PDP-7 on which Unix was developed had less processing power than a typical toaster oven does today. A typical personal computer of 2003 is more than a million times more powerful. Yet the main tasks for which Unix computers are used — software development, document processing, communications — have changed comparatively little. Where is all that additional power going? What new behaviors of the machine and the humans using it does it support?

3. (Comprehension) Three pivotal events in Unix history took place just before 1970, around 1980, and just after 1990. Describe each in one or two sentences of your own words.

4. (Application) Character-input speeds on the earliest Unix terminals were 110 bits per second, five hundred times slower than a 56K modem connection today. Identify a characteristic of Unix's interface design that might be traceable to this speed difference.

5. (Application) Unix has been used to drive printing terminals, serial-line video terminals with character-only displays, and the bit-mapped graphics displays of today's PCs and workstations. Read the stty(1) manual page. Identify a feature that was useful on serial-line terminals but is not useful for modern graphics displays. For extra credit, identify a feature that was useful on printing terminals but not on video terminals.

6. (Application) Unix uses a LF (ASCII 10) as a line terminator. Most Internet protocols use CR-LF (ASCII 10 and 13) as a line terminator. The difference occasionally causes problems. Relate this to the chapter's description of two separate cultures before 1980. Which convention do you suppose the PDP-10 used?

7. (Analysis) Corporations prefer high profit margins to low profit margins, and are reluctant to sacrifice margin in the short term for long-term gain. How does this help explain the failure of Sun and other workstation manufacturers to cope with the onslaught of the PC?

8. (Analysis) Monopolies are hostile to technical innovation that touches their core business, because it can undercut the value of their investments by making the equipment and techniques that they own obsolete. Unix was born within the Bell Telephone monopoly. Why was that allowed to happen?

9. (Synthesis) In 1991, two separate attempts to write or port Unix for PCs were launched without the authors learning of each other. Also in that year, the World Wide Web was invented. If a project like Linux or 386BSD were to launch today, do you think authors of potentially overlapping projects would be more likely or would be less likely than in 1991 to hear about it? Justify your answer. For extra credit, argue both sides.

10. (Synthesis) Expensive machines mean that software development requires large concentrations of capital and big organizations to manage resources. Does this mean the Unix wars of 1984-1994 were inevitable? How might they have been avoided if inexpensive but powerful PCs had been available ten years sooner?

11. (Evaluation) Read enough of *Why Software Should Be Free* [http://www.gnu.org/philosophy/shouldbefree.html] so that you feel you have the gist of the argument. Next, do likewise with the three case documents at the Open Source Initiative's advocacy page [http://www.opensource.org/advocacy/]. Compare these arguments. Which is more logically convincing to you? Which is more emotionally appealing? Would your answers change if you were the CEO of a Fortune 500 corporation?

12. (Evaluation) The narrative of the History chapter is half driven by technological trends and half by larger-than-life individual figures like Ken Thompson and Linus Torvalds. Human beings seem to need or create heroes when they have revolutions, but the heroes ride tremendous waves created by technology and economics. Choose one of the names in this chapter; decide, on the evidence available to you, to what extent you think the man created the time versus the time creating the man. Explain. Expect to be graded not on your conclusion but on the quality and depth of your explanation.

Chapter 3. Contrasts

Suggested Exercises

Have students use the Web to find documentation on a historical timesharing operating system. Here are some plausible candidates; give students both the short and long names for use as web-search keys.

- CTSS (Compatible Time Sharing System).
- DTSS (Dartmouth Time Sharing System).
- GCOS (General Comprehensive Operating System).
- ITS (Incompatible Timesharing System).
- MPE (Multi-Programming Executive).
- MTS (Michigan Terminal System).
- TOPS-10
- TOPS-20

Ask them to describe one of these systems in the terms developed in this chapter. As all but MPE died before Web archiving was common, information on them is incomplete. Students will have to make inferences and justify them; this is part of the exercise.

Concepts and definitions

operating system	Software which mediates between user-visible applications and the hardware, managing resources such as memory and disk and scheduling processor time.

timesharing	A timesharing operating system is one that supports multiple simultaneous users.
multitasking	The ability to support multiple processes, each one of which normally looks to users as though it is running continuously.
multiuser	The ability to support not just multiple tasks but multiple users, with uers owning resources such as files so that other users cannot modify or even read them.
memory-management unit (MMU)	Hardware that creates internal firewalls between processes in memory so they cannot modify each other.
CLI	Command-line interface. The kind in which which users type text commands to a general-purpose interpreter (as opposed to a special-purpose forms-based interface).
GUI	Graphical User Interface. The kind in which which users communicate their intentions mainly through gesturing with a pointing device (such as a mouse) in a visual, graphics-intensive interface.
server operating system	An operating system that emphasises multi-user support and long continuous uptimes.
client operating sytem	An operating system that emphasizes single-user operation with a GUI.

Knowledge questions

1. In what decade was VMS first released?

   The 1970s.

2. In what kind of machine would you be most likely to find OS/2 running today?

An automated teller machine.

3. In Windows NT, a process spawn takes approximately how long?

   A tenth of a second.

4. Name the application area for which BeOS was specialized.

   Multimedia, especially sound and video manipulation.

5. What is the unifying concept of MVS?

   Everything is a batch job.

6. Which of the operating systems described in this chapter originated on the IBM 7094 mainframe?

   VM/CMS.

7. What year did Linux originate?

   1991.

8. Name two open-source Unixes other than Linux.

   FreeBSD, NetBSD, OpenBSD, and Darwin are all acceptable answers. There might be more in the future; if a student's answer looks wrong, challenge him or her to back it up with a web reference.

Essay questions

1. (Comprehension) Over time, the cost ratios between hardware and human programmer-hours have changed drastically as hardware costs have plummeted. Discuss the distinction between server and client operating systems in this light.

2. (Comprehension) Explain, in your own words, the concept of “internal boundaries” in operating systems, and its practical impact. What kinds of problems do internal boundaries prevent?

3. (Comprehension) From the eight case studies in Chapter 3, give three operating systems with a predominantly command-line user interface and three with a predominantly graphical interface.

4. (Application) Suggest a relationship between the resurgence in Unix after 1993 or so with the emergence of a mass-market Internet Service Provider industry around the same time.

5. (Application) Chapter 3 classifies timesharing operating systems according to the following list of traits:

   1. Presence or absence of a unifying idea
   2. Multitasking capability
   3. Cooperating processes
   4. Internal boundaries
   5. Textual vs, binary system file formats
   6. Preferred user interface style
   7. Intended audience
   8. Entry barriers to development

Suppose we consider operating systems designed for a single user without networking. Name two traits that can drop out of the discussion. Explain why.

6. (Application) Consider operating systems designed for *embedded* use — that is, for controlling toasters or machine tools or aviation electronics. Can any traits drop out of the discussion? Explain.

7. (Analysis) The book asserts “In a world of cheap hardware, the presence or absence of [a casual-programming] community is an increasingly important factor in whether an operating system is long-term viable at all.” It is implied that the presence of a casual-programming community was less important to an operating system’s health back when hardware was expensive. Explain why this might be so. Can you find analogies in other kinds of engineering? Consider, for example, the design of automobiles or boats.

8. (Analysis) Why is it important for an operating system to have a unifying idea? The book mentioned that because Windows NT lacks one, its system APIs tend to be unstable over time. Explain this. Predict at least one other consequence of not having a unifying idea.

9. (Synthesis) Linux is moving away from using a periodic clock interrupt, towards having its timeslice allocation driven more by external interrupts such as keyboard, disk, and network device I/O notifications. What if there were no clock interrupt? What sorts of problems might arise?

10. (Synthesis) Suppose inexpensive and ubiquitous networking had developed before the personal computer rather than after it. Which of the personal-computer operating systems described in this chapter (MacOS, OS/2, Windows NT, BeOS, Linux) would have to be most different from its present design? Which would be least affected? Explain your reasoning.

11. (Evaluation) One effect of Unix’s use of flat textual files for system databases is that most administrative manipulations can be done using a text editor rather than through many different special-purpose tools. Contrast the effect of having a fatal bug in your text editor under Unix with having fatal bugs in some percentage of your special-purpose administration tools under other operating systems. Which would you rather cope with? How does that judgment change (if at all) assuming you know in advance that you have several editors handy?

12. (Evaluation) The price of application-development tools for a proprietary operating system is primarily set not by their development cost but by the vendor’s business strategy for maximizing revenue from the operating system. The general rule for all commodities (not just software) is that high prices collect a higher margin from a smaller market; low prices capture a smaller margin but encourage market growth. Pick a pair of operating systems for which the vendor strategy is different

and compare the effectiveness of the high-cost/high-margin strategy with the low-cost/low-margin strategy. Put this in context with other trends discussed in the History chapter.

Chapter 4. Modularity

Suggested Exercises

Have the students read the documentation for two or three Unix libraries from one of the manual pages in section 3 (libraries) and apply the criteria of this chapter. Is the library strictly compact or semi-compact? Is it orthogonal? If it fails to have either of these properties, why? In assigning the libraries, each student or group of students should get at least one that you judge to be strictly compact or semi-compact and one that you judge to be non-compact.

Concepts and definitions

API	Application Programming Interface. The entry points into a module, the set of procedure calls and data structures that it exposes to the outside world.
modularity	Software is modular when it is divided into pieces that have narrow interfaces with each other, separating function and holding down the global complexity of the system.
Brooks's Law	Adding programmers to a late project makes it later. More generally, costs and error rates rise as the square of the number of programmers on a project.
compactness	The property that a design can fit inside a human being's head.
orthogonality	A software design has orthogonality when each entry point or command does exactly one thing.
glue logic	Code that glues modules together. Bugs tend to collect in glue.
object orientation	A style of programming in which data and the functions that act on it are bundled together into self-contained objects.

Knowledge questions

1. The ‘sweet spot’ in module size is where?

   Between 200 and 400 logical lines, or 400 to 800 physical ones.

2. For which common programming language has the designer admitted that he does not expect any one programmer to get it all?

   C++.

3. Name the minicomputer on which the C language grew up.

   C.

4. The name of the core library of the GIMP application is:

   libgimp.

5. What does SPOT stand for?

   Single Point Of Truth.

6. Changing the organization of your code without changing the core algorithms is called what?

   Refactoring.

7. Name the dual of diff(1).

patch(1)

8. The analog of the SPOT rule for data structures is:

“No junk, no confusion”

Essay questions

1. (Comprehension) Explain why compactness is an important property of software.

2. (Comprehension) Restate the concept of orthogonality in your own words.

3. (Comprehension) By default, ls(1) columnates its output. Is this a compactness violation, an orthogonality violation, or a SPOT violation? Explain.

4. (Application) Explain how the Hatton curve might be applied to predict an optimum maintainable size for pages on a Web site, and why that might be an appropriate thing to do.

5. (Application) List what you think are the seven most commonly used entry points of stdio(3), the Unix standard I/O library. Is this library semi-compact? If not, what other entries would you have to include to span 85% of your normal use of it?

6. (Application) Under Unix, emitting a newline to a terminal or terminal emulator will generally take the cursor to the beginning of the next line on the screen (scrolling the screen up if necessary); the kernel does character translation on the fly to ensure this. Under most other operating systems, the same operation requires a CR (to take the cursor to start of the current line) and LF (to move it down one line). Is the Unix design an orthogonality violation? Does your answer change depending on whether the terminal is a printing device or a video display? Does your answer change if the terminal has move-to-end-of-line and *line starve* (reverse-line-feed) control codes to complement CR and LF? Explain your reasoning.

7. (Analysis) Consider the **enum** feature in C. What are its benefits and costs? Would the design of C approach Saint-Exupéry’s perfection more closely if it were removed, or not? Explain your answer.

8. (Analysis) Brooks's Law predicts that costs and error rates rise as the square of the number of programmers on a project. N squared is the number of (undirectional) communications channels between N points. Connect these facts. Explain what cost and overhead is associated with each new communication channel.

9. (Synthesis) Compare the **rehash** command of csh(1) with **hash** in bash(1) by reading their manual pages. Identify the critical difference that makes one of these straightfoward (if obscure) but the other a recipe for headaches. Generalize from this.

10. (Synthesis) One of the modularity questions we ask is "Do your data structures have more than seven members?". Does data structure consisting of a single eight-element array of integers have an interface that we should suspect of leaking its owning module's internals? Does it make a difference whether all the locations in the array are used in a the same way or whether different locations in it have different semantics? Generalize.

11. (Evaluation) We've discussed several reasons for being a bit warier of heavily object-oriented designs than is fashionable outside the Unix world. Give three. Rank them in importance. Discuss the difference bwtween their effect on machine behavior and on buman (developer) behavior; explain which is more important, and why.

12. (Evaluation) Psychologist Mihaly Csikszentmihalyi has popularized the concept of "flow state", a form of intense concentration some people enter during creative activity. Investigate this concept using the Web or other available resources. Do you enter a flow state when you write programs? Have you ever been "in the zone" while playing a video game? Discuss the connections among "flow state", starting from zero, and the Saint-Exupéry quote about perfection,

Chapter 5. Textuality

Suggested Exercises

Ask students to match the following applications with the data file metaformat most appropriate for each:

- A simple contact list of names and phone numbers: DSV style.
- A tree of text descriptions associated with subsystems and parts in a computer: XML.
- A list of lightbulb jokes: Fortune-cookie format.
- A Unix mailbox: RFC 822 format.
- A list of bibliography entries with Title, Author, Publisher, ISBN fields and an associated text synopsis or abstract: RFC-822 record-jar format.

Concepts and definitions

serialization	The act of transforming a complex data structure with multiple fiekds (possibly including address pointers) into a representation that can be saved to disk and restored without loss of information.
textual vs. binary	Textual formats and protocols contain only printable characters and are designed to be manipulated with ordinary text tools. Binary protocols use opaque binary encodings and have to be handled with special single-purpose tools.
RFC 822	The Internet standard that describes the format of Internet mail (revised as RFC 2822).
metaformat	A set of syntax rules that defines a family of related data-file formats, each with its own semantics.

XML	Extensible Markup Language, a metaformat for documents and datafiles related to HTML.
application protocol	A set of rules that defines how two software applications can communicate data and/or commands.

Knowledge questions

1. An alternate term for serialization is:

   Unmarshalling.

2. When you feel the urge to design a complex binary file format, or a complex binary application protocol:

   Lie down until the feeling passes.

3. Password-file format uses what character as a field delimiter?

   A colon (:).

4. The standard describing how to embed binary enclosures in RFC 822 messages is called:

   MIME (Multipurpose Internet Media Extension)

5. What is the Unix-standard introducer for comments in data files?

   The pound sign (#).

6. Name the three classic examples of Internet protocols described in this chapter.

   SMTP, POP3, and IMAP.

7. A PNG file is a sequence of self-describing chunks. What is the principal advantage of this organization?

   No need for an overall version number.

8. What is the least surprising thing for the character sequence `"\b"` in data file to mean?

   That the user wants an embdded backspace to appear at that point in the in-memory version of the string or field.

Essay questions

1. (Comprehension) Explain, in your own words, why PNG is a binary format rather than a textual one.

2. (Comprehension) Identify three distinct potential problems with passing binary data structures between dissimilar machines.

3. (Comprehension) Describe the DSV metaformat in your own words.

4. (Application) The result from a CDDB/freedb.org query is a plain text message listing key=value pairs, all text strings. Go to freedb.org and perform such a query for your favorite album. Look at the output and design an extension to report the album's record label.

5. (Application) Describe a rule for marking or prefixing lines in text parts of a record-jar file so that text lines consisting of `%%\n` that occur can be distinguished from intentional record delimiters. How does your rule avoid creating a collision of its own with text lines that were entered with the mark or prefix already attached?

6. (Application) Sketch an extension of the `.newsrc` that can associate property-value pairs with each group (such as a priority property that says how the group should be ranked in the user's to-read list). The design should have the property that when the attribute list of a newsgroup is empty, the data associated with the newsgroup is identical to what it would be in the unextended design. How will you handle edge cases like property values with embedded spaces?

7. (Analysis) Investigate the Base64 Content-Transfer-Encoding in RFC 2045. This incurs a 33% space penalty over unencoded binaries. Any non-empty file on a Unix system has a physical disk allocation of one disk block minimum; on modern Unixes this is usually 4K bytes. You have been asked to design a storage format for square, bit-mapped icons of a fixed size with pixels one byte deep. Ignoring the fixed cost of headers, what is the largest size that will allow you to use base64 without wasting any disk space?>

8. (Analysis) In the password-file example, we suggested about an 8% improvement in space efficiency from bit-packing the numeric fields and the password string. We computed this percentage with the program given below, by counting the number of characters the example would have if the numeric fields were single binary bytes and the password strings were 7/8ths their current size, rounded up to the nearest byte; then we looked at the ratio of sizes. Apply this program to the password file on your nearest Unix system. Is the ratio larger or smaller? Develop an explanation for this. What does it suggest about the general case?

```
#!/bin/env/python
# Compute shrinkage percentage from binary-packing a password file.
import operator, sys, string
data = map(lambda x: string.split(x, ':'), sys.stdin.readlines())
basecount = reduce(operator.add,
       map(lambda x: len(x[0])+len(x[4])+len(x[5])+len(x[6]),data))
unpacked  = basecount + reduce(operator.add,
       map(lambda x: len(x[1])+len(x[2])+len(x[3]), data))
packed = basecount + reduce(operator.add,
       map(lambda x: int((len(x[1])+0.5)*(7.0/8.0)) + 1 + 1, data))
print "%2.2f%%" % (((unpacked - packed) * 1.0 / unpacked) * 100,)
```

9. (Synthesis) Design a textual equivalent of the utmp(5) file format. Make sure the text format you invent expresses 100% of the information in the binary format. Use the simplest of the metaformats described in this chapter that will do the job, and follow all the design rules for that metaformat. Expect to be graded on the tastefulness of the result as well as its utility.

10. (Synthesis) Invent a notation for base-2 backslash escapes that is compatible with the traditional backslash conventions.

11. (Evaluation) Explain why the notation you invented in the previous exercise is unlikely to pass into general use. What existing convention(s) make it redundant?

12. (Evaluation) POP uses an end delimiter to mark message boundaries; IMAP gives an explicit length of message instead. One choice (delimiter or length field) lays more cost on the client system, and one on the server. Which is which? Bearing in mind that computation has become cheaper more rapidly than has storage, and assuming that servers are usually faster machines than clients, pick one of these as best practice. Justify your conclusion.

Chapter 6. Transparency

Suggested Exercises

Match the following applications or facilities with their descriptions:

- audacity — audio file editor
- freeciv — multiplayer strategy game
- fetchmail — network gateway program
- smbclient — SAMBA fileshare browser
- kmail — GUI mail client
- gcc — C compiler
- sng — textualizes PNG graphics
- terminfo — terminal capability database
- psql — database browser
- dig — DNS record textualizer

Concepts and definitions

transparency	A program is transparent when it is possible to form a simple mental model of its behavior that is actually predictive for all or most cases, because you can see through the machinery to what is actually going on.

discoverability	Code (or a user interface) is discoverable when it is strutured to help you learn about itself.
elegance	The combination of power and simplicity.
lossless conversion	A data translation that is perfectly information-preserving.
textualizer	A program that losslessly converts between some complex or opaque binary representation of data and a simple textual one.

Knowledge questions

1. The fetchmail application handles which two common remote-mail protocols?

   POP3 and IMAP.

2. The status bar in kmail is modeled on that of what other program?

   Netscape or Mozilla.

3. The syntax of SNG resembles that of what Web standard?

   CSS (Cascading Style Sheets.),

4. Terminfo replaced what earlier and now obsolete database?

   Termcap.

5. GCC stands for:

The Gnu C Compiler. (Gnu Compiler Collection should also be an acceptable answer, though it's not given in the book.)

6. The name of the program used to compile terminal descriptions into the terminfo database.

   tic, the TermInfo Compiler.

7. The smbclient(1) browser program is associated with what project?

   SAMBA.

8. The intermediate textual files used by the components of GCC are not normally interesting to human beings, but are nevertheless extremely useful for what activity?

   Regression testing.

Essay questions

1. (Comprehension) Explain, in your own words, the difference between transparency and discoverability.

2. (Comprehension) Identify an example of elegant design (combining simplicity, power, and transparency) in a tool or common household object. Hint: Think about the artifacts that you most enjoy using.

3. (Comprehension) Describe three qualities or practices that incease the maintainability of code.

4. (Application) Any binary file can be textualized by running it through a hex- or octal-dump utility like od(1); the result can be edited with a text editor and then converted back to binary with a simple Perl or Python program. Explain why this is not a helpful observation.

5. (Application) The TODO file in the *sng* source distribution sketches a proposed XML syntax to supplement or replace the CSS-like one now implemented. Read this proposal. Would adopting

it increase, decrease or not affect the degree of transparency in a system of scripts being used to modify PNG images? Give your reasoning.

6. (Application) Explain how you would write a regression tester for a program that generates PNG images, using SNG and *diff*.

7. (Analysis) The text states: “Syntax errors in an SNG file can be corrected by hand without requiring specialized editors that might refuse to load a damaged PNG image.” Explain this. What makes the image-loading code in a graphics editor so much more sensitive?

8. (Analysis) Investigate the terminfo binary-entry directory on your system to find out if it has an entry for the Lear-Siegler ADM-3A (do not look in the source text of the terminfo file). Now forget the answer and think about what you did while looking. There is at least one good effect of the initial-letter directories in the terminfo binary-entry tree that has nothing to do with improving lookup performance and everything to do with human convenience. Is there a general insight here about transparency of data structures? Explain.

9. (Synthesis) The user is clued into what Audacity’s Play, Stop, and Pause buttons mean by multiple cues — shape (corresponding to the conventions used on a lot of consumer electronics) and color (like the lights on a stoplight) among them. Generalize: is there a general guideline for promoting interface discoverability here?

10. (Synthesis) Textualizers, by definition, emit and accept plain text. The interactive interface of a browser, on the other hand, may be CLI or GUI. Explain why CLI is usually a better choice, and why Web browsers are an exception to this rule. Generalize from this.

11. (Evaluation) The *fetchmail* `-v` dumps everything about a session except actual message texts: transactions with the mailserver, transactions with the local SMTP listener, fetchmail’s own status messages. Often this is more information than you want. Some programs verbose option which takes a bitmask or feature-set argument that selectively enables and disables specific kinds of reporting. Would it have been better to design the `-v` of *fetchmail* this way? How might that have affected the working habits of people testing the program? Discuss the tradeoffs.

12. (Evaluation) One of the questions this chapter suggests you ask yourself about a design is “Are there a handful of prominent data structures or a single global scoreboard that captures the high-level state of the system?” When the answer to this question is yes, it becomes easier for a human being to hold a mental model of the implementation in his or her head. Is this feature affected by whether these core data structures are objects in the OO sense, that is whether they include procedural methods as well as state data? Discuss the complexity implications.

Chapter 7. Multiprogramming

Suggested Exercises

Match each technique to the application that uses it:

1. mutt — shelling out
2. pic2graph — pipeline
3. rblsmptd — Bernstein chaining
4. scp — slave process
5. PostgreSQL — sockets
6. Apache — shared memory

Concepts and definitions

multiprogramming	Splitting programs up into multiple cooperating processes.
threads	Multiple comcurrent processes sharing the same memory-address space.
	signals

Asynchronous notifications sent to a process which trigger handler code.

sockets	Bidirectional byte streams connecting two processes.

Knowledge questions

1. Which signal is frequently used to tell daemons they should reread their configuration files?

   SIGHUP.

2. Your SMTP listener normally watches for transactions on what port?

   25.

3. Name the environment variable normally used to control which editor your mailer shells out to.

   EDITOR

4. What is the name of the proxy program used to direct editing requests to a running instance of Emacs?

   emacsclient.

5. Name the mail-transport-agent package that uses Bernstein chaining extensively.

   qmail.

6. Name the shell variable that expanss to the PID of the current process.

   $$.

7. Which lineage of signals introduced “reliable” (non-resetting) signals.

   BSD.

8. Which signal is associated with asynchronous I/O?

   SIGIO.

9. Name the two programs called by *pic2graph*.

   groff(1) and convert(1).

10. Which signal is normally used to gracefully terminate a program?

    SIGTERM.

Essay questions

1. (Comprehension) Explain, in your own words, the difference between a shellout and a slave process.

2. (Comprehension) Describe “ontology creep”. Explain the kinds of problems it causes.

3. (Comprehension) Explain, in your own words, the concept of peer-to-peer communication.

4. (Application) Suppose you want to invoke a Web browser to view the content attached to an URL that has been generated by some application (say, as a way of accessing on-line help). Is this a good candidate for a shellout, a master/slave relationship, or some form of peer-to-peer IPC? Explain.

5. (Application) Find a way to improve the *pic2graph* example script by replacing one tempfile write with a pipe bar. Hint: Consider the - option of *convert*.

6. (Application) Starting from the **ls | wc** example and the sort(1) manual page, write a pipeline that gives a list of files sorted by line count. There are two different combinations of sort(1) and wc(1) options you might use; the difference in the output is that one version of it will include character/word/line counts for every file, while the other includes line counts only. Explain which you think is more likely to be generally useful.

7. (Analysis) The description of how PostgreSQL works in the text omits a detail — the user does not actually start up a PostgreSQL client directly, but rather starts a launcher program that starts the client. Construct an explanation. (Hint: recall the discussion of the setuid bit and privilege-dropping.)

8. (Analysis) Show that “yes/no/maybe” property of socket I/O (that is, the recipient may be using bytes in a message that the transmitting machine got a timeout for) is not specific to TCP/IP, but general to any protocol that used transmits followed by acknowledgment or timeout.

9. (Synthesis) Sketch a design for breaking *fetchmail* into fetcher and injector programs using Bernstein chaining rather than pipelines. Describe at least one problem with this approach.

10. (Synthesis) Relate Ken Arnold’s explanation of the invention of SIGUSR1 and SIGUSR2 to the Rule of Extensibility. Generalize; what sorts of bad code can you expect to accumulate around a design that needs extension hooks but doesn’t have them?

11. (Evaluation) Recent implementations of bc(1) are monolithic, rather than partitioned to use dc(1) as a slave process. How do these approaches compare along the axes of performance, code complexity, and features? Propose a plausible reason for the monolithic organization or modern *bc*.

12. (Evaluation) Doug McIlroy reveals that the real reason for the proliferation of IPC methods in older Unixes is that they are the debris from Bell Labs turf wars. If you are not already familiar with the history of GNOME and KDE, read The Story of the GNOME project [http://primates.ximian.com/~miguel/gnome-history.html] to discover why GNOME was formed. Are these two cases parallel? Compare them and discuss.

Chapter 8. Minilanguages

Suggested Exercises

Investigate the following languages and place them on the taxonomy diagram (Figure 8.1): HTML, the sendmail.cf configuration language, Scheme, inittab(5) format, TeX, terminfo(5).

Concepts and definitions

declarative minlanguage	A minilanguage in which the actions being described are implied by a declaration of relationships or constraints.
imperative minilanguge	A language that has explicit actions and control structures specifying how to act in some data.
macro language	A language speciealized for describing text-to-text transformation by expansion of named macros.
regular expression	A notation for describing patterns in text.

Knowledge questions

1. Name the programmable editor that was the ancestor of Emacs.

   TECO.

2. Which minilanguage is used to descrobe transformations of XML trees?

XSLT.

3. What regular-expression operator describes one or more repeats of a pattern element?

   The + suffix. In some variants the suffix {1,} will do this.

4. Which two troff preprocessors require *pic* because they emit *pic* commands in order to do their jobs?

   chem and *grap*.

5. Name the earlier language that PostScript most resembles.

   FORTH.

6. *m4* is used to hide the ugliness of what famously hideous configuration language?

   The sendmail.cf configuration language.

7. What is the variant of regular expressions used for filename expansion by the shell called?

   Glob expressions.

8. Glade markup is an application of what data file metaformat?

   XML.

9. *pic* diagrams have to be surrounded by what pair of delimiters?

.PS and .PE.

10. ECMA-262 is the European standard for what language?

    JavaScript.

Essay questions

1. (Comprehension) Explain the difference between a data file format and a minilanguage in your own words.

2. (Comprehension) Given that both languages are Turing-complete, explain what makes Perl or Python “more general” on the taxonomy diagram than Emacs Lisp or JavaScript.

3. (Comprehension) In the Glade example, identify the part of the XML that tells the GTK library it should construct a button.

4. (Application) Write a grep(1) invocation that can extract the second of three colon-separated fields from a line of text. Now read the cut(1) manual page. Why and where would you use a regexp rather than this utility?

5. (Application) The following line of shell uses bc(1) to give the shell variable pi the value of π (the mathematical constant) to ten decimal places: **pi=$(echo "scale=10; 4*a(1)" | bc -l)**. Under what circumstances might this be a better design than supplying a literal value of π?

6. (Application) Using what you learned from the source listing for the taxonomy diagram, write a *pic* program that draws three adjacent ellipses marked “Door #1”, “Door #2”, and “Door #3”; verify that you have correct results.

7. (Analysis) There is an arrow from left to right on the language taxonomy diagram marked “increasing loopiness”. What trait is being indicated here?

8. (Analysis) The text states that the main advantage of PostScript is that PostScript descriptions of images are smaller than the corresponding bitmaps. This is an advantage for machines. Identify the corresponding advantage for humans.

9. (Synthesis) Use what you learned in Exercise 6 to write a script that, for a given command-line argument N, draws N adjacent ellipses marked “Door #1” through “Door #N”. This program will assert a relationship among a bunch of ellipses and leave placement up to the implementation. Working from the description of PostScript in the text, consider the kind of program you would have had to write to accomplish the same thing in PostScript. Would that have been simpler or more difficult? Why?

10. (Synthesis) Generalize the result of the previous exercise. What is it about box diagrams that makes it easier (or more difficult) to manipulate them by changing relationships among the parts rather than by modifying drawing commands? Can you identify other organizations of data for which this is so? (Hint: look at your desktop.)

11. (Evaluation) Examine, in the *pic* source of the taxonomy diagram, the statements that place text labels. For other statements that place objects, there is a leading keyword such as **line** or **box** that describes the object type. Would the language be improved by a leading **text** keyword? Evaluate the pros and cons in terms of readability, compactness, expressiveness, esthetic feel, and other relevant criteria. Expect to be graded on the quality of your argument rather than the specific answer you arrive at.

12. (Evaluation) Regexp notation is not orthogonal; for example, the regexps aa*, a+, and a{1,} all mean the same thing. While this is partly historical accident, it is also true that people designing regexp variants have been willing to discard orthogonality to gain concision and expressiveness. Evaluate this tradeoff. Is there something specific about the way regexps are that makes orthogonality less important than in a general-purpose language, or concise expression more imortant?

Chapter 9. Generation

Suggested Exercises

Concepts and definitions

data-driven programming	Code in which the logic of the program is implied by the shape of a data structure.
code generation	Using programs to write programs (starting from higher-level specifications which are not themselves code).

Knowledge questions

1. Name at least two different statistical spam filters.

   popfile, spambayes, and bogofilter are acceptable.

2.

3.

4.

5.

6.

7.

8.

9.

10.

Essay questions

1. (Comprehension) Describe, in your own words, the difference between object-oriented and data-driven programming.

2. (Comprehension)

3. (Comprehension)

4. (Application)

5. (Application)

6. (Application)

7. (Analysis)

8. (Analysis)

9. (Synthesis)

10. (Synthesis)

11. (Evaluation)

12. (Evaluation)