Open Source Software in Libraries: A Workshop

by Eric Lease Morgan

Open Source Software in Libraries: A Workshop

by Eric Lease Morgan

This text/handout is a part of a hands-on workshop for teaching people in libraries about open source software.

For possibly more up-to-date information about the workshop, see:
http://infomotions.com/musings/ossnlibraries-workshop/
[http://infomotions.com/musings/ossnlibraries-workshop/] .

Table of Contents

Chapter 1. Introduction


[./ossnlibraries-workshop.png]

Purpose and scope of this text/workshop

This text is a part of a hands-on workshop intended to describe and illustrate open source software and its techniques to small groups of librarians. Given this text, the accompanying set of software, and reasonable access to a (Unix) computer, the student should be able to read the essays, work through the exercises, and become familiar with open source software especially as it pertains to libraries.

I make no bones about it, this text is the combination of previous essays I've written about open source software as well as a couple of other newer items. For example, the second chapter is the opening chapter I wrote for a LITA Guide in 2002 ("Open Source Software for Libraries," in Karen Coyle, ed., Open Source Software for Libraries: An Open Source for Libraries: Chicago: American Library Association, 2002 pg. 7-18.). The third chapter comparing open source software, gift cultures, and librarianship was originally formally published as a book review for Information Technology and Libraries (volume 19, number 2, March 2000). The chapter on open source software indexers is definitely getting old. It was presented at the O'Reilly Open Source Convention, San Diego, CA July 23-27, 2001. The following section is built from the content of a 2001 American Libraries Association Annual Conference presentation. The new materials are embodied in the list of selected software and the hands-on activities.

I believe open source software is more about building communities and less about computer programs. It is more about making the world a better place and less about personal profit. Allow me to explain.

I have been giving away my software ever since Steve Cisler welcomed me into the Apple Library Of Tomorrow (ALOT) folds in the very late 1980's. Through my associations with Steve and ALOT I came to write a book about Macintosh-based HTTP servers as well as an AppleScript-based CGI script called email.cgi in 1994.

This simple little script was originally developed for two purposes. First and foremost it was intended to demonstrate how to write an AppleScript Common Gateway Interface (CGI) application. Second, it was intended to fill a gap in the Web browsers of the time, namely the inability of MacWeb to support mailto URL's. Since then the script has evolved into an application taking the contents of an HTML form, formatting it, and sending the results to one or more email addresses. It works very much like a C program called cgiemail. As TCP

utilities have evolved over the years so has email.cgi, and to this date I still get requests for technical support from all over the world, but almost invariably the messages start out something like this. "Thank you so very much for email.cgi. It is a wonderful program, but..." That's okay. The program works and it has helped many people in many ways -- more ways than I am able to count because the vast majority of people never contacted me personally.

As I was bringing this workbook together I thought about Steve Cisler again, and I remembered a conference Apple Computer sponsored in 1995 called Ties That Bind: Converging Communities. (A pretty bad travel log documenting my experiences at this conference is available at http://infomotions.com/travel/ties-that-bind-95/.) In the conference we shared and discussed ideas about community and the ways technology can help make communities happen. In between a session Cisler displayed the original piece of art that became the motif for the conference. He noted that he got the painting in Australia some time the previous year. He liked it for its simplicity and connectivity. The painting is acrylic, approximately 1' 6" X 2" 6", and is composed of many simple dots of color.

The image at the top of the page is that piece of art, and it is significant today. It too is "a lot" (all puns intended) like open source software and the "the Unix way." The value of open source software is measured in terms of its simplicity and connectivity. The simpler and more connective the software, the more it is valued. The Unix way is a philosophy of computing. It posits that a computer program will take some input, do some processing, and provide some output. There is very little human interface to these sorts of programs because they get their input from a thing called standard input (STDIN) and send the output to a thing called standard output (STDOUT). If errors occur, errors are sent to standard error (STERR). Since the applications are expected to get their input from STDIN and send it to STOUT it is possible to string many together to create a working application. Connectivity. Such a design philosophy allows tiny programs to focus on one thing, and one thing only. Simplicity. This modular approach allows for the creation of new applications by adding or deleting older modules from the string.

The motif brought to my attention by Cisler is a lot like stringing together open source software applications. Each individual dot does not do a whole lot on its own, but strung together they form a pattern. The pattern's whole is greater than the sum of its parts. This is true of communities as well. Individuals bring something to the community, and the community is made better for the contribution. The open source community exists because of individuals. These individuals have particular strengths (and weaknesses). As people add what they can to the community, the community is strengthened. The rewards for these contributions are rarely monetary. Instead, the contributions are paid for with respect. People who give freely of themselves and their time are rewarded by the community as experts whose opinions are to be taken seriously. True, participation in open source software activities does not always put food on the table, but neither do other community-based activities our society values to one degree or another such as participation in community theater, helping out at the local soup kitchen, being involved in church activities, picking up litter, giving directions to a stranger, supporting charities, participating in fund-raisers, etc. Open source software is about communities, communities that have been easier to create with the advent of globally networked computers. As described later, it is about "scratching an itch" to solve a problem, but it is also about giving "freely" to the community in the hopes that the community will be better off for it in the end.

A few years after writing email.cgi, I participated in another application called MyLibrary. This portal application grew out of a set of focus group interviews where faculty of the NC State University said they were suffering from information overload. In late 1997, when these interviews were taking place, services like My Yahoo, My Excite, My Netscape, and My DejaNews were making their initial appearance. In the Digital Library Initiatives Department, where I worked Keith Morgan and Doris Sigl, we thought a similar appli-

cation based on library content (bibliographic databases, electronic journals, and Internet resources) organized by subjects (disciplines) might prove to be a possible solution to the information overload problem. By prescribing sets of resources to specific groups of people we (the Libraries) could offer focused content as well as provide access to the complete world of available information.

Since I relinquished my copyrights to the University and the software has been distributed under the GNU Public License the software has been downloaded about 350 times, mostly from academic libraries. The specific number of active developers is unknown, but many institutions who have downloaded the software have used it as a model for their own purposes. In most cases these institutions have taken the system's database structure and experimented with various interfaces and alternative services. Such institutions include, but are not limited to the University of Michigan, the California Digital Library, Wheaton College, Los Alamos Laboratory, Lund University (Sweden), the University of Cattaneo (Italy), and the University of New Brunswick. Numerous presentations have been given about MyLibrary including venues such as Harvard University, Oxford University, the Alberta Library, the Canadian Library Association, the ACRL Annual Meeting, and ASIS.

As I see it, there are three or four impediments restricting greater success of the project: system I/O, database restructuring, and technical expertise. MyLibrary is essentially a database application with a Web front-end. In order to distribute content data must be saved in the database. The question then is, "How will the data be entered?" Right now it must be done by content providers (librarians), but the effort is tedious and as the number of bibliographic databases and electronic journals grow so does the tedium. Lately I have been experimenting with the use of RDF as an import/export mechanism. By relying on some sort of XML standard the system will be able to divorce itself from any particular database application such as an OPAC and the system will be more able to share its data with other portal applications such as uPortal, My Netscape, or O'Reilly's Meerkat through something like RSS. Yet, the problem still remains, "Who is going to do the work?" This is a staffing issue, not necessarily a technical one.

In order to facilitate the needs a wider audience, the underlying database needs to be restructured. For example, the databases contains tables for bibliographic databases, electronic journals, and "reference shelf" items. Each of the items in these tables are classified using a set of controlled vocabulary terms called disciplines. Many institutions want to create alternative data types such as images, associations, or Internet resources. Presently, do accomplish this task oodles of code must be duplicated bloating the underlying Perl module. Instead a new table needs to be created to contain a new controlled vocabulary called "formats". Once this table is created all the information resources could be collapsed into a single table and classified with the new controlled vocabulary as well as the disciplines. Furthermore, a third controlled vocabulary -- intended audience -- could be created so the resources could be classified even further. Given such a structure the system could be more exact when it comes to initially prescribing resources and allowing users to customize their selections. Again, the real problem here is not necessarily technical but intellectual. Librarians make judgments about resources in terms of the resource's aboutness, intended audience, and format all the time but rarely on such a large scale, systematic basis. Our present cataloging methods do not accommodate this sort of analysis, and how will such analysis get institutionalized in our libraries?

The comparitavly low level of technical expertise in libraries is also a barrier to wider acceptance of the system. MyLibrary runs. It doesn't crash nor hang. It does not output garbage data. It works as advertised, but to install the program initially requires technical expertise beyond the scope of most libraries. It requires the installation of a database program. MySQL is the current favorite, but there are all sort of things that can go wrong with a MySQL installation. Similarly, MyLibrary is written in Perl. Installing Perl

from source usually requires answering a host of questions about your computer's environment, and in all nine or ten years of compiling Perl I still don't know what some of those questions mean and I simply go with the defaults. Then there are all the Perl modules MyLibrary requires. They are a real pain to install, and unless you have done these sorts of installs before the process can be quite overwhelming. In short, getting MyLibrary installed is not like the Microsoft wizard process; you have to know a lot about your host computer before you can even get it up and running and most libraries do not employ enough people with this sort of expertise to make the process comfortable.

This workbook brings together much of my experience with open source software. It describes sets of successful open source software projects and tries to enumerate the qualities of successful project. The workbook has been in the hopes people will read it, give the exercises a whirl, learn from the experience, and share their newly acquired expertise with the world at large. Through this process I hope we can make the world we live in just a little bit better place. Idealist? Maybe. A worthy goal? Definitely.

Chapter 2. Open Source Software in Libraries

Introduction

This guide is an introduction to open source software in libraries, with descriptions of a variety of software packages and successful library projects. But before we get to the software itself, I want to describe the principles and techniques of open source software (OSS) and explain why I advocate the adoption of OSS in the implementation of library services and collections.

As you will see, there are many shared principles between OSS and librarianship, especially the free and equal access to information. Because of the freedom we gain with the use of OSS is it possible to have greater control over the ways computers function and therefore greater control over how libraries operate. Anybody who works with computers on a daily basis can contribute to OSS because things like information architecture, usability testing, documentation, and staffing are key skills required for successful projects, and these skills are inherent in the people who use computers as a primary tool in their work. The implementation of OSS in libraries represents a method for improving library services and collections.

What is OSS

OSS is both a philosophy and a process. As a philosophy it describes the intended use of software and methods for its distribution. Depending on your perspective, the concept of OSS is a relatively new idea being only four or five years old. On the other hand, the GNU Software Project -- a project advocating the distribution of "free" software -- has been operational since the mid '80's. Consequently, the ideas behind OSS have been around longer than you may think. It begins when a man named Richard Stallman worked for MIT in an environment where software was shared. In the mid '80's Stallman resigned from MIT to begin developing the GNU -- a software project intended to create an operating system much like Unix. (GNU is pronounced "guh-NEW" and is a recursive acronym for GNU's Not Unix.) His desire was to create "free" software, but the term "free" should be equated with freedom, and as such people who use "free" software should be:

1. free to run the software for any purpose
2. free to modify the software to suit their needs
3. free to redistribute of the software gratis or for a fee
4. free to distribute modified versions of the software

Put another way the term "free" should be equated with the Latin word "liberat" meaning to liberate, and not necessarily "gratis" meaning without return made or expected. In the words of Stallman, we should "think of 'free' as in 'free speech,' not as in 'free beer.'"[1]

Fast forward to the late '90's after Linus Torvalds successfully develops Linux, a "free" operating system on par with any commercial Unix distribution. Fast forward to the late '90's when globally networked computers are an every day reality and the .com boom is booming. There you will find the birth of the term "open source" and it is used to describe how software is licensed:

- the license shall not restrict any party from selling or giving away soft-

ware

- the program shall include source code and must allow distribution of the code
- the license shall allow modifications and derived work of the software
- the license may restrict redistribution only if patches (fixes) are included
- the license may not discriminate against any person or group of persons
- the license may not restrict how the software is used
- the rights attached to the program must apply to all whom the software is redistributed
- the license must not be specific to a product
- the license must not contaminate other software by place restrictions on it [2]

Techniques for developing and implementing OSS

OSS is also a process for the creation and maintenance of software. This is not a formalized process, but rather a process of convention with common characteristics between software projects. First and for most, the developer of a software project almost always is trying to solve a specific computer problem commonly called "scratching an itch." The developer realizes other people may have the same problem(s), and consequently the developer makes the project's source code available on the 'Net in the hopes other people can use it too.

If there seems to be a common need for the software, a mailing list is usually created to facilitate communication, and the list is hopefully archived for future reference. Since the software is almost always in a state of flux, developers need some sort of version control software to help manage the project's components. The most common version control software is called CVS (Concurrent Versions System). Co-developers then "hack away" at the project adding features they desire and/or fixing bugs of previous releases. As these features and fixes are created the source code's modifications, in the form of "diff" files -- specialized files explicitly listing the differences between two sets of programming code -- are sent back to the project's leader. The leader examines the diff files, assesses their value, and decides whether or not to integrate them into the master archive. The cycle then begins anew. Much of a project's success relies on the primary developer's ability to foster communication and a sense of community around a project. Once accomplished the "two heads are better then one" philosophy takes effect and the project matures.

Writing computer programs is only one part of the software development. Software development also requires things such as usability testing, documentation, beta-testing, and a knowledge of staff issues. Consequently, in any environment where computers are used on a daily basis are places where the techniques of OSS can be practiced. Knowledge of computer programming is not necessary. In fact, a lack of computer programming is desireable. You do not have to know how to write computer programs in order to participate in OSS development.

Anybody who uses computers on a daily basis can help develop OSS. For example, you can be a beta-tester who tries to use the software and finds its faults. You can write documentation instructing people how to use the software. You can conduct usability tests against the software discovering how easy the

software is to use or not use, and how it meets people's expectations. If computer software is intended to make our lives easier, you can evaluate the use of the software and see what sorts of things can be eliminated or how resources can be reallocated in order to run operations more efficiently. All of these things have nothing to do with computer programming, but rather, the use of computers in a work place.

OSS Compared to Librarianship

One the most definitive sets of writings describing OSS is Eric Raymond's The Cathedral and the Bazaar.[3] These texts, available online as well as in book form, compare and contrast the software development processes of monolithic organizations (Cathedrals) with the software processes of less structured, more organic collections of "hackers" (Bazaars).[4] The book describes the environment of free software and tries to explain why some programers are willing to give away the products of their labors. It describes the "hacker milieu" as a "gift culture":

> Gift cultures are adaptations not to scarcity but to abundance. They arise in populations that do not have significant material scarcity problems with survival goods. We can observe gift cultures in action among aboriginal cultures living in ecozones with mild climates and abundant food. We can also observe them in certain strata of our own society, especially in show business and among the very wealthy.[5]

Raymond alludes to the definition of "gift cultures", but not enough to satisfy my curiosity. The literature, more often than not, refers to information about "gift exchange" and "gift economies" as opposed to "gift cultures." Probably one of the earliest and more comprehensive studies of gift exchange was written by Marcell Mauss.[6] In his analysis he says gifts, with their three obligations of giving, receiving, and repaying, are in aspects of almost all societies. The process of gift giving strengthens cooperation, competitiveness, and antagonism. It reveals itself in religious, legal, moral, economic, aesthetic, morphological, and mythological aspects of life.[7]

As Gregory states, for the industrial capitalist economies, gifts are nothing but presents or things given, and "that is all that needs to be said on the matter." Ironically for economists, gifts have value and consequently have implications for commodity exchange.[8] He goes on to review studies about gift giving from an anthropological view, studies focusing on tribal communities of various American indians, cultures from New Guinea and Melanesia, and even ancient Roman, Hindu, and Germanic societies:

The key to understanding gift giving is apprehension of the fact that things in tribal economics are produced by non-alienated labor. This creates a special bond between a producer and his/her product, a bond that is broken in a capitalistic society based on alienated wage-labor.[9]

Ingold, in "Introduction To Social Life" echoes many of the things summarized by Gregory when he states that industrialization is concerned:

exclusively with the dynamics of commodity production. ... Clearly in non-industrial societies, where these conditions do not obtain, the significance of work will be very different. For one thing, people retain control over their own capacity to work and over other productive means, and their activities are carried on in the context of their relationships with kin and community. Indeed their work may have the strengthening or regeneration of these relationships as its principle objective.[10]

In short, the exchange of gifts forges relationships between partners and emphasizes qualitative as opposed to quantitative terms. The producer of the

product (or service) takes a personal interest in production, and when the product is given away as a gift it is difficult to quantify the value of the item. Therefore the items exchanged are of a less tangible nature such as obligations, promises, respect, and interpersonal relationships.

As I read Raymond and others I continually saw similarities between librarianship and gift cultures, and therefore similarities between librarianship and OSS development. While the summaries outlined above do not necessarily mention the "abundance" alluded to by Raymond, the existence of abundance is more than mere speculation. Potlatch, a ceremonial feast of the American Indians of the northwest coast marked by the host's lavish distribution of gifts or sometimes destruction of property to demonstrate wealth and generosity with the expectation of eventual reciprocation, is an excellent example.

Libraries have an abundance of data and information. I won't go into whether or not they have an abundance of knowledge or wisdom of the ages. That is another essay. Libraries do not exchange this data and information for money; you don't have to have your credit card ready as you leave the door. Libraries don't accept checks. Instead the exchange is much less tangible. First of all, based on my experience, most librarians just take pride in their ability to collect, organize, and disseminate data and information in an effective manner. They are curious. They enjoy learning things for learning's things sake. It is a sort of Platonic end in itself. Librarians, generally speaking, just like what they do and they certainly aren't in it for the money. You won't get rich by becoming a librarian.

Even free information is not without financial costs. Information requires time and energy to create, collect, and share, but when an information exchange does take place, it is usually intangible, not monetary, in nature. Information is intangible. It is difficult to assign information a monetary value, especially in a digital environment where it can be duplicated effortlessly:

An exchange process is a process whereby two or more individuals (or groups) exchange goods or services for items of value. In Library Land, one of these individuals is almost always a librarian. The other individuals include tax payers, students, faculty, or in the case of special libraries, fellow employees. The items of value are information and information services exchanged for a perception of worth -- a rating valuing the services rendered. This perception of worth, a highly intangible and difficult thing to measure, is something the user of library services "pays", not to libraries and librarians, but to administrators and decision-makers. Ultimately, these payments manifest themselves as tax dollars or other administrative support. As the perception of worth decreases so do tax dollars and support. [11]

Therefore when information exchanges take place in libraries librarians hope their clientele will support the goals of the library to administrators when issues of funding arise. Librarians believe that "free" information ("think free speech, not free beer") will improve society. It will allow people to grow spiritually and intellectually. It will improve humankind's situation in the world. Libraries are only perceived as beneficial when they give away this data and information. That is their purpose, and they, generally speaking, do this without regards to fees or tangible exchanges.

In many ways I believe OSS development, as articulated by Raymond, is very similar to the principles of librarianship. First and foremost with the idea of sharing information. Both camps put a premium on open access. Both camps are gift cultures and gain reputation by the amount of "stuff" they give away. What people do with the information, whether it be source code or journal articles, is up to them. Both camps hope the shared information will be used to improve our place in the world. Just as Jefferson's informed public is a necessity for democracy, OSS is necessary for the improvement of computer applications.

Second, human interactions are a necessary part of the mixture in both librarianship and open source development. Open source development requires people skills by source code maintainers. It requires an understanding of the problem the computer application is trying to solve, and the maintainer must assimilate patches with the application. Similarly, librarians understand that information seeking behavior is a human process. While databases and many "digital libraries" house information, these collections are really "data stores" and are only manifested as information after the assignment of value are given to the data and inter-relations between datum are created.

Third, it has been stated that open source development will remove the necessity for programers. Yet Raymond posits that no such thing will happen. If anything, there will an increased need for programmers. Similarly, many librarians feared the advent of the Web because they believed their jobs would be in jeopardy. Ironically, librarianship is flowering under new rubrics such as information architects and knowledge managers.

OSS also works in a sort of peer review environment. As Raymond states, "Given enough eyeballs, all bugs are shallow." Since the source code to OSS is available for anybody to read, it is possible to examine exactly how the software works. When a program is written and a bug manifests itself, there are many people who can look at the program, see what it is doing, and offer suggestions or fixes.

Instead of relying on marketing hype to promote an application, OSS relies on its ability to satisfy particular itches to gain prominence. The better a piece of software works, the more people are likely to use it. User endorsements are usually the way OSS is promoted. The good pieces of software float to the top because they are used the most often. The ones that are poorly written or do not satisfy enough itches sink to the bottom.

In a peer review process many people look at an article and evaluate its validity. During this evaluation process the reviews point out deficiencies in the article and suggest improvements. The reviewers are usually anonymous but authoritative. The evaluation of OSS often works in the same vein. Software is evaluated by self-selected reviewers. These people examine all aspects of the application from the underlying data structures, to the way the data is manipulated, to the user interface and functionality, to the documentation. These people then offer suggestions and fixes to the application in an effort to enhance and improve it.

Some people may remember the "homegrown" integrated library systems developed in the '70's and '80's, and these same people may wonder how OSS is different from those humble beginnings. There are two distinct differences. The first is the present-day existence of the Internet. This global network of computers enables people to communicate over much greater distances and it is much less expensive than twenty-five years ago. Consequently, developers are not as isolated as they once were, and the flow of ideas travels more easily between developers -- people who are trying to scratch that itch. Yes, there were telephone lines and modems but the processes for using them was not as seemlessly integrated into the computing environment (and there were always long-distance communications charges to contend with.[12])

Second, the state of computer technology and its availability has dramatically increased in the past twenty-five years. Twenty-five years ago computers, especially the sorts of computers used for large-scale library operations, were almost always physically large, extremely expensive, remote devices whose access was limited to a group of few specialized individuals. Now-a-days, the computers on most people's desktops have enough RAM, CPU horsepower, and disk space to support the college campus of twenty-five years ago.[13]

In short, the OSS development process is not like the homegrown library systems of the past simply because there are more people with more computers who are able to examine and explore the possibilities of solving more computing

problems. In the times of the homegrown systems people were more isolated in their development efforts and more limited in their choice of computing hardware and software resources.

Prominent OSS Packages

There are quite a number of mainstream OSS applications. Many of these applications literally run the Internet or are used for back-end support. The Apache Project is one of the more notable (www.apache.org). Apache is a World Wide Web (HTTP) server. It started out its life in the mid '90's as NCSA's httpd application, the Web server beneath the first graphical Web browser. The name for the application -- Apache -- is a play on words. It has nothing to do with indians. Instead, in an effort to write a more modular computer program, the original httpd application was rewritten as a set of parts, or patches, and consequently the application is called "a patchy server." Few experts would doubt the popularity of the Apache server. According to Netcraft, more HTTP servers are Apache HTTP server than any other kind. [14]

MySQL is a popular relational database application. It is very often used to support database-driven websites. It adhears to the SQL standard while adding a number of features of its own (as does Oracle and other database vendors). MySQL is known for its speed and stability. The canonical address for MySQL is www.mysql.org.

Sendmail is an email (SMTP) server used on the vast majority of Unix computers. This application, developed quite a number of years ago is responsible for trafficing much of the email messages sent throughout the world. Sendmail is a good example of an application supported by both a commercial institution as well as a non-profit organization. There is a free version of sendmail, complete with source code, as well as a commercial version that comes with formal support. See www.sendmail.org.

BIND is an acronym for the Berkeley Internet Name Domain, a program converting Internet Protocol (IP) numbers, such as 17.112.144.32 into human-readable names such as www.apple.com. It is a sort like an old fashioned switchboard operator associating telephone numbers with the telephones in people's homes. BIND is supported by the Internet Software Consortium at www.isc.org.

Perl is a programming language written by Larry Wall in the late '80's. It too runs much of the Internet since it is used as the language of many common gateway interface (CGI) scripts of the internet. Wall originally created Perl to help him do systems administration task, but the language worked so well others adopted it and it has grown significantly. Perl is supported at www.perl.com.

Linux is the most familiar OSS application. This program is really an operating system -- a program directly responsible converting human-readable commands into computer (machine) language. It is the software that really makes computers run. Linux was originally conceived by Linus Torvols in the late '80's because he wanted to run a Unix-sort of operating system on Intel-based computer. Linux is becoming increasingly popular with many information technology (IT) professionals as an alternative to Windows-based server applications or proprietary versions of Unix. See www.linux.org.

State of OSS in Libraries

Daniel Chudnov has been the library profession's OSS evangelist for the past three or four years. He is also the original author of the open source program jake (jointly administered knowledge environment). Chudnov has done a lot to raise the awareness of OSS in libraries. To that end he and others help maintain a website called OSS4Lib (www.oss4lib.org). The site lists library-related applications including applications for document delivery, Z39.50 clients and servers, systems to manage collections, MARC record readers and

writers, integrated library system, and systems to read and write bibliographies. For more information visit OSS4Lib and subscribe to the mailing list.

The state of OSS in libraries is more than sets of computer programs. It also includes the environment where the software is intended to be used -- a socio-economic infrastructure. Any computing problem can be roughly divided into 20% technology issues and 80% people issues. It is this 80% of the problem that concerns us here. Given the current networked environment, the affinity of OSS development to librarianship, and the sorts of projects enumerated above what can the library profession do to best take advantage of the currently available OSS? I posed this question to the OSS4Lib mailing list in April and May of 2000 and it generated a lively discussion. [15] A number of themes presented themselves, each of which are elaborated upon below.

National leadership

One of the strongest themes was the need for a national leader. It was first articulated by David Dorman as the OSLN (Open Source Library Network). Karen Coyle and Aaron Trehab elaborated on this idea by suggesting organizations such as ALA/LITA, the DLF, OCLC, or RLG help fund and facilitate methods for providing credibility, publicity, stability, and coordination to library-based OSS projects.

Mainstreaming, workshops, and training

Along theses same lines was the expressed desire for the mainstreaming of OSS articulated by Carol Erkens, Rachel Cheng, and Peter Schlumpf. This mainstreaming process would include presentations, workshops, and training sessions on local, regional, and national levels. These activities would describe and demonstrate open sources software for libraries. They would enumerate the advantages and disadvantages of open sources software. They would provide extensive instructions on the staffing, installation, and maintenance issue of OSS.

Usability and packaging

In its present state, open sources software is much like microcomputer computing of the '70's as stated by Blake Carver. It is very much a build it yourself enterprise; the systems are not very usable when it comes to installation. This point was echoed by Cheng who recently helped facilitate a NERCOMP workshop on OSS. Peter Schulmpf points to the need for easier installation methods so maintainers of the system can focus on managing content and not software. Using OSS should not be like owning an automobile in the 1920's. "I shouldn't necessarily need to know how to fix it in order to make it go."

Economic viability

OSS needs to be demonstrated as an economically viable method of supporting software and systems. This was pointed out by Eric Schnell and David Dorman. Libraries have spent a lot of time, effort, and money on resource sharing. Why not pool these same resources together to create software satisfying our professional needs? OSS is not like the "homegrown" systems. Spaghetti code and GOTO statements should be a thing of the past. More importantly, a globally networked computer environment provides a means of sharing expertise in a manner not feasible twenty-five years ago. We need to demonstrate to administrators and funding sources that money spent developing software empowers our collective whole. It is an investment in personnel and infrastructure. OSS is not a fad, yet is will not necessarily replace commercial software. On the other hand, OSS offers opportunities not necessarily available from the commercial sector.

Redefining the ILS

There are many open source library application available today. Each satisfies a particular need. Maybe each of these individual applications can be brought together into a collective, synergistic whole as described by Jeremy Frumkin and we could redefine the integrated library system. Presently our ILS's manage things like books pretty well. With the addition of 856 fields in MARC records they are beginning to assist in the management of networked resources, but libraries are more than books and networked resources. Libraries are about services too: reserves, reading lists, bibliographies, reader advisory services of many types, current awareness, reference, etc. Maybe the existing OSS can be glued together to form something more holistic resulting in a sum greater than its parts. This is also an opportunity, as described by Schnell, for vendors to step in and provide such integration including installation, documentation, and training.

Open source data

OSS relates to data as well as systems as described by Krichel. The globally networked computer environment allows us to share data as well as software. Why not selectively feed URL's to Internet spiders to create our own, subject-specific indexes? Why not institutionalize services like the Open Directory Project or build on the strength of INFOMINE to share records in a manner similar to the manner of OCLC?

Conclusion and next steps

This essay has described what OSS is and it compared OSS to the principles of librarianship. The balance of the book details particular systems of OSS for libraries. After reading this book I hope you go away understanding at least one thing. OSS provides the means for the profession to take greater control over the ways computers are used in libraries. OSS is free, but it is free in the same way freedom exists in a democracy. With freedom comes choice. With freedom comes the ability to manifest change. At the same time, freedom comes at a price, and that price is responsibility. OSS puts its users in direct control of computer operations, and this control costs in terms of accountability. When the software breaks down, you will be responsible for fixing it. Fortunately, there is a large network at your disposal, the Internet, not to mention the creator of the software who has the same problems you do and has most likely previously addressed the same problem. Open source provides the means to say, "We are not limited by our licensed software because we have the ability to modify the software to meet our own ends." Instead of blaming vendors for supporting bad software, instead of confusing the issues with contractual agreements and spending tens of thousands of dollars a year for services poorly rendered, OSS offers an alternative. Be realistic. OSS is free, but not without costs.

This being the case, what sorts of things need to happen for OSS to become a more viable computing option in libraries? What are the next steps? The steps fall into two categories: 1) making people more aware of OSS and 2) improving the characteristics of OSS.

Librarians need to become more aware of the options OSS provides. This can be done in a number of ways. For example, a formal study analyzing the desirability and feasibility of libraries making a formal commitment to OSS might demonstrate to other libraries the benefits of OSS. Library boards and directors need feel comfortable commiting funds to OSS installation and development, but before doing so the boards and directors need to know what OSS is and how its principles can be applied in libraries. By mentoring existing librarians to become more computer literate the concepts of OSS will become easier to understand. Similarly, by mentoring librarians to be more aware of the ways of administration these same librarians will have more authority to make decisions and direct energies to OSS development. All librarians should not be afraid of the idea of open sources software because they think computer programming experience is necessary. There is much more to software development

than writing computer programs. Simple training exercises will also make more people aware of the potential of open sources software. Finally, communication -- testimonials -- will help disseminate the successes, as well as failures, of OSS.

OSS itself needs to be improved. The installation processes of OSS are not as simple as the installation procedures of commercial software. This is area that needs improvement, and if done, fewer people would be intimidated by the installation process. Additionally, there are opportunities for commercial institutions to support OSS. These institutions, like Red Hat or O'Reilly & Associates, could provide services installing, documenting, and trouble shooting OSS. These institutions would not be selling the software itself, but services surrounding the software.

The principles of OSS of very similar to the principles of librarianship. Let's take advantage of these principles and use them to take more control of over our computing environments.

Notes

1. The ideas behind GNU software and its definition as articulated by Richard Stallman can be found at http://www.gnu.org/philosophy/free-sw.html. Accessed April 25, 2002.

2. Much of the preceeding section was derived from Dave Bretthaur's excellent article, "OSS: A History" in Information Technology and Libraries 21(1) March, 2002. pg. 3-10.

3. The Cathedral and the Bazaar is also available online at http://www.tuxedo.org/~esr/writings/cathedral-bazaar/. Accessed April 25, 2002.

4. It is important to distinguish here the difference between a "hacker" and a "cracker". As defined by Raymond, a hacker is person who writes computer programs because they are "scratching an itch" -- trying to solve a particular computer problem. This definition is contrasted with the term "cracker" denoting a person who maliciously tries to break computer systems. In Raymond's eyes, hacking is a noble art, cracking is immoral. It is unfortunate, the distinction between hacking and cracking seems to have been lost on the general population.

5. Raymond, E.S., The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary. 1st ed. 1999, [Sebastopol, CA]: O'Reilly. pg. 99.

6. Mauss, M., The gift; forms and functions of exchange in archaic societies. The Norton library, N378. 1967, New York: Norton.

7. Lukes, S., Mauss, Marcel, in International encyclopedia of the social sciences, D.L. Sills, Editor. 1968, Macmillan: [New York] volume 10, pg. 80.

8. Gregory, C.A, "Gifts" in Eatwell, J., et al., The New Palgrave : a dictionary of economics. 1987, New York: Stockton Press. volume 3, pg. 524.

9. Ibid.

10. Ingold, T., Introduction To Social Life, in Companion encyclopedia of anthropology, T. Ingold, Editor. 1994, Routledge: London ; New York. p. 747.

11. Morgan, E.L., "Marketing Future Libraries", http://www.infomotions.com/musings/marketing/. Accessed April 25, 2002.

12. As an interesting aside, read "Stalking the wily hacker" by Clifford Stoll

in the Communications of the ACM May 1988 31(5) pg. 484. The essay describes how Clifford tracked a hacker via a 75 cent error in his telephone bill. It is on the Web in many places. Try http://eserver.org/cyber/stoll2.txt. Accessed April 25, 2002

13. It is believed a past chairman of IBM, Thomas Watson, said in 1943, "I think there is a world market for maybe five computers."

14. See http://www.netcraft.com for more information. Accessed April 25, 2002.

15. An archive of the oss4lib mailing list is available at this URL http://www.geocrawler.com/lists/3/SourceForge/6067/0/. Accessed April 25, 2002.

Chapter 3. Gift Cultures, Librarianship, and Open Source Software Development

Gift Cultures, Librarianship, and Open Source Software Development

This short essay examines more closely the concept of a "gift culture" and how it may or may not be related to librarianship. After this examination and with a few qualifications, I still believe my judgments about open source software and librarianship are true. Open source software development and librarianship have a number of similarities -- both are examples of gift cultures.

I have recently been reading a book about open source software development by Eric Raymond. [1] The book describes the environment of free software and tries to explain why some programers are willing to give away the products of their labors. It describes the "hacker milieu" as a "gift culture":

Gift cultures are adaptations not to scarcity but to abundance. They arise in populations that do not have significant material scarcity problems with survival goods. We can observe gift cultures in action among aboriginal cultures living in ecozones with mild climates and abundant food. We can also observe them in certain strata of our own society, especially in show business and among the very wealthy. [2]

Raymond alludes to the definition of "gift cultures", but not enough to satisfy my curiosity. Being the good librarian, I was off to the reference department for more specific answers. More often than not, I found information about "gift exchange" and "gift economies" as opposed to "gift cultures." (Yes, I did look on the Internet but found little.)

Probably one of the earliest and more comprehensive studies of gift exchange was written by Marcell Mauss. [3] In his analysis he says gifts, with their three obligations of giving, receiving, and repaying, are in aspects of almost all societies. The process of gift giving strengthens cooperation, competitiveness, and antagonism. It reveals itself in religious, legal, moral, economic, aesthetic, morphological, and mythological aspects of life. [4]

As Gregory states, for the industrial capitalist economies, gifts are nothing but presents or things given, and "that is all that needs to be said on the matter." Ironically for economists, gifts have value and consequently have implications for commodity exchange. [5] He goes on to review studies about gift giving from an anthropological view, studies focusing on tribal communities of various American indians, cultures from New Guinea and Melanesia, and even ancient Roman, Hindu, and Germanic societies:

The key to understanding gift giving is apprehension of the fact that things in tribal economics are produced by non-alienated labor. This creates a special bond between a producer and his/her product, a bond that is broken in a capitalistic society based on alienated wage-labor.[6]

Ingold, in "Introduction To Social Life" echoes many of the things summarized by Gregory when he states that industrialization is concerned:

exclusively with the dynamics of commodity production. ... Clearly in non-industrial societies, where these conditions do not obtain, the significance of work will be very different. For one thing, people retain control over their own capacity to work and over other productive means, and their activities are carried on in the context of their relationships with kin and commu-

nity. Indeed their work may have the strengthening or regeneration of these relationships as its principle objective. [7]

In short, the exchange of gifts forges relationships between partners and emphasizes qualitative as opposed to quantitative terms. The producer of the product (or service) takes a personal interest in production, and when the product is given away as a gift it is difficult to quantify the value of the item. Therefore the items exchanged are of a less tangible nature such as obligations, promises, respect, and interpersonal relationships.

As I read Raymond and others I continually saw similarities between librarianship and gift cultures, and therefore similarities between librarianship and open source software development. While the summaries outlined above do not necessarily mention the "abundance" alluded to by Raymond, the existence of abundance is more than mere speculation. Potlatch, "a ceremonial feast of the American Indians of the northwest coast marked by the host's lavish distribution of gifts or sometimes destruction of property to demonstrate wealth and generosity with the expectation of eventual reciprocation", is an excellent example. [8]

Libraries have an abundance of data and information. (I won't go into whether or not they have an abundance of knowledge or wisdom of the ages. That is another essay.) Libraries do not exchange this data and information for money; you don't have to have your credit card ready as you leave the door. Libraries don't accept checks. Instead the exchange is much less tangible. First of all, based on my experience, most librarians just take pride in their ability to collect, organize, and disseminate data and information in an effective manner. They are curious. They enjoy learning things for learning's things sake. It is a sort of Platonic end in itself. Librarians, generally speaking, just like what they do and they certainly aren't in it for the money. You won't get rich by becoming a librarian.

Information is not free. It requires time and energy to create, collect, and share, but when an information exchange does take place, it is usually intangible, not monetary, in nature. Information is intangible. It is difficult to assign it a monetary value, especially in a digital environment where it can be duplicated effortlessly:

An exchange process is a process whereby two or more individuals (or groups) exchange goods or services for items of value. In Library Land, one of these individuals is almost always a librarian. The other individuals include tax payers, students, faculty, or in the case of special libraries, fellow employees. The items of value are information and information services exchanged for a perception of worth -- a rating valuing the services rendered. This perception of worth, a highly intangible and difficult thing to measure, is something the user of library services "pays", not to libraries and librarians, but to administrators and decision-makers. Ultimately, these payments manifest themselves as tax dollars or other administrative support. As the perception of worth decreases so do tax dollars and support. [9]

Therefore when information exchanges take place in libraries librarians hope their clientele will support the goals of the library to administrators when issues of funding arise. Librarians believe that "free" information ("think free speech, not free beer") will improve society. It will allow people to grow spiritually and intellectually. It will improve humankind's situation in the world. Libraries are only perceived as beneficial when they give away this data and information. That is their purpose, and they, generally speaking, do this without regards to fees or tangible exchanges.

In many ways I believe open source software development, as articulated by Raymond, is very similar to the principles of librarianship. First and foremost with the idea of sharing information. Both camps put a premium on open access. Both camps are gift cultures and gain reputation by the amount of "stuff" they give away. What people do with the information, whether it be

source code or journal articles, is up to them. Both camps hope the shared information will be used to improve our place in the world. Just as Jefferson's informed public is a necessity for democracy, open source software is necessary for the improvement of computer applications.

Second, human interactions are a necessary part of the mixture in both librarianship and open source development. Open source development requires people skills by source code maintainers. It requires an understanding of the problem the computer application is trying to solve, and the maintainer must assimilate patches with the application. Similarly, librarians understand that information seeking behavior is a human process. While databases and many "digital libraries" house information, these collections are really "data stores" and are only manifested as information after the assignment of value are given to the data and inter-relations between datum are created.

Third, it has been stated that open source development will remove the necessity for programers. Yet Raymond posits that no such thing will happen. If anything, there will an increased need for programmers. Similarly, many librarians feared the advent of the Web because they believed their jobs would be in jeopardy. Ironically, librarianship is flowering under new rubrics such as information architects and knowledge managers.

It has also been brought to my attention by Kevin Clarke (kevin_clarke@unc.edu) that both institutions use peer-review:

Your cultural take (gift culture) on "open source" is interesting. I've been mostly thinking in material terms but you are right, I think, in your assessment. One thing you didn't mention is that, like academic librarians, open source folks participate in a peer-review type process.

All of this is happening because of an information economy. It sure is an exciting time to be a librarian, especially a librarian who can build relational databases and program on a Unix computer.

Acknowledgements

Thank you to Art Rhyno (arhyno@server.uwindsor.ca) who encouraged me to post the original version of this text.

Notes

1. Raymond, E.S., The cathedral and the bazaar : musings on Linux and open source by an accidental revolutionary. 1st ed. 1999, [Sebastopol, CA]: O'Reilly.

2. Ibid. pg. 99.

3. Mauss, M., The gift; forms and functions of exchange in archaic societies. The Norton library, N378. 1967, New York: Norton.

4. Lukes, S., Mauss, Marcel, in International encyclopedia of the social sciences, D.L. Sills, Editor. 1968, Macmillan: [New York] volume 10, pg. 80.

5. Gregory, C.A, "Gifts" in Eatwell, J., et al., The New Palgrave : a dictionary of economics. 1987, New York: Stockton Press. volume 3, pg. 524.

6. Ibid.

7. Ingold, T., Introduction To Social Life, in Companion encyclopedia of anthropology, T. Ingold, Editor. 1994, Routledge: London ; New York. p. 747.

8. Merriam-Webter Online Dictionary, http://search.eb.com/cgi-bin/dictionary?va=potlatch

9. Morgan, E.L., Marketing Future Libraries,
http://www.lib.ncsu.edu/staff/morgan/cil/marketing/

Chapter 4. Comparing Open Source Indexers

Abstract

This text compares and contrasts the features and functionality of various open source indexers: freeWAIS-sf, Harvest, Ht://Dig, Isite/Isearch, MPS, SWISH, WebGlimpse, and Yaz/Zebra. As the size of information systems increase so does the necessity of providing searchable interfaces to the underlying data. Indexing content and implementing an HTML form to search the index is one way to accomplish this goal, but all indexers are not created equal. This case study enumerates the pluses and minuses of various open source indexers currently available and makes recommendations on which indexer to use for what purposes. Finally, this case study will make readers aware that good search interfaces alone to not make for good information systems. Good information systems also require consistently applied subject analysis and well structured data.

Indexers

Below are a few paragraphs about each of the indexers reviewed here. They are listed in alphabetical order.

freeWAIS-sf

Of the indexes reviewed here, freeWAIS-sf is by far the grand daddy of the crowd, and the predecessor Isite/Isearch, SWISH, and MPS. Yet, freeWAIS-sf is not really the oldest indexer because it owes its existence to WAIS originally developed by Brewster Kahle of Thinking Machines, Inc. as long ago as 1991 or 1992.

FreeWAIS-sf supports a bevy of indexing types. For example, it can easily index Unix mbox files, text files where records are delimited by blank lines, HTML files, as well as others. Sections of these text files can be associated with fields for field searching through the creation "format files" -- configuration files made up of regular expressions. After data has been indexed it can be made accessible through a CGI interface called SFgate, but the interface relies on a Perl module, WAIS.pm, which is very difficult to compile. The interface supports lots o' search features including field searching, nested queries, right-hand truncation, thesauri, multiple-database searching, and Boolean logic.

This indexer represents aging code. Not because it doesn't work, but because as new incarnations of operating systems evolve freeWAIS-sf get harder and harder to install. After many trials and tribulations, I have been able to get it to compile and install on RedHat Linux, and I have found it most useful for indexing two types of data: archived email and public domain electronic texts. For example, by indexing my archived email I can do free text searches against the archives and return names, subject lines, and ultimately the email messages (plus any attachments). This has been very helpful in my personal work. Using the "para" indexing type I have been able to index a small collection of public domain literature and provide a mechanism to search one or more of these texts simultaneously for things like "slave" to identify paragraphs from the collection.

Harvest

Harvest was originally funded by a federal grant in 1995 at the University of Arizona. It is essentially made up of two components: gatherers and brokers. Given sets of one or more URLs, gatherers crawl local and/or remote file sys-

tems for content and create surrogate files in a format called SOIF. After one or more of the SOIF collections have been created they can be federated by a broker, an application indexing them and makes them available though a Web interface.

The Harvest system assumes the data being indexed is ephemeral. Consequently, index items become "stale", are automatically removed from retrieval, and need to be refreshed on a regular basis. This is considered a feature, but if your content does not change very often it is more a nuisance than a benefit.

Harvest is not very difficult to compile and install. It comes with a decent shell script allowing you to set up rudimentary gatherers and brokers. Configuration is done through the editing of various text files outlining how output is to be displayed. The system comes with a Web interface for administrating the brokers. If your indexed content is consistently structured and includes META tags, then it is possible to output very meaningful search results that include abstracts, subject headings, or just about any other fields defined in the META tags of your HTML documents.

The real strength of the Harvest system lies in its gathering functions. Ideally system administrators are intended to create multiple gatherers. These gatherers are designed to be federated by one or more brokers. If everybody were to index their content and make it available via a gatherer, then a few brokers can be created collecting the content of the gatherers to produce subject- or population-specific indexes, but alas, this was a dream that came to fruition.

Ht://Dig

This is nice little indexer, but just doesn't have the features of some of the other available distributions. Configuring the application for compilation is not too tricky, but unless you set paths correctly you may create a few broken links. Like SWISH, to index your data you feed the application a configuration file and it then creates gobs of data. Many indexes can be created and they then have to be combined into a single database for searching. Not too hard.

The indexer supports Boolean queries, but not phrase searching. It can apply an automatic stemming algorithm, but upon doing so you might give the unsuspecting user information overload. The search engine does not support field searching, and a rather annoying thing is that the indexer does not remove duplicates. Consequently, index.html files almost always appear twice in search results. On the other hand, one nice thing Ht://Dig does do that the other engines don't do (except WebGlimpse) is highlight query terms in a short blurb (a pseudo-abstract) of the search results. Ht://Dig is a simple tool. Considering the complexity of some of the other tools reviewed here, I might rank this one as #2 after SWISH.

Isite/Isearch

Isite/Isearch is one of the very first implementations based on the WAIS code. Like Yaz/Zebra, it is intended to support the Z39.50 information retrieval protocol. Like freeWAIS (and unlike Yaz/Zebra) it supports a number of file formats for indexing. Unfortunately, Isite/Isearch no longer seems to be supported and the documentation is weak. While it comes with a CGI interface and is easily installed, the user interface is difficult to understand and needs a lot of tweaking before it can be called usable by today's standards. If you require Z39.50 compliance and for some reason Yaz/Zebra does not work for you, then give Isite/Isearch a whirl.

MPS

MPS seems to be the zippiest of the indexers reviewed here. It can create more data in a shorter period of time than all of the other indexers. Unlike the

other indexers MPS divides the indexing process into two parts: parser and indexer. The indexer accepts what is called a "structured index stream", a specialized format for indexing. By structuring the input the indexer expects it is possible to write output files from your favorite database application and have the content of your database indexed and searchable by MPS. You are not limited to indexing the content of databases with MPS. Since it too was originally based on the WAIS code it indexes many other data types such as mbox files, files where records are delimited by blank lines (paragraphs), as well as a number of MIME types (RTF, TIFF, PDF, HTML, SOIF, etc.). Like many of the WAIS derivatives, it can search multiple indexes simultaneously, supports a variant of the Z39.50 protocol, and a wide range of search syntax.

MPS also comes with a Perl API and an example CGI interface. The Perl API comes with the barest of documentation, but the CGI script is quite extensive. One of the neatest features of the example CGI interface is its ability to allow users to save and delete searches against the indexes for processing later. For example, if this feature is turned on, then a user first logs into the system. As the user searches the system their queries are stored to the local file system. The user then has the option of deleting one or more of these queries. Later, when the user returns to the system they have the option of executing one or more of the saved searches. These searches can even be designed to run on a regular basis and the results sent via email to the user. This feature is good for data that changes regularly over time such a news feeds, mailing list archives, etc.

MPS has a lot going for it. If it were able to extract and index the META tags of HTML documents, and if the structured index stream as well as the Perl API were better documented, then this indexer/search engine would ranking higher on the list.

SWISH

SWISH is currently my favorite indexer. Originally written by Kevin Hughes (who is also the original author of hypermail), this software is a model of simplicity. To get it to work for you all that needs to be done is to download, unpack, configure, compile, edit the configuration file, and feed the file to the application. A single binary and a single configuration file is used for both indexing and searching. The indexer supports Web crawling. The resulting indexes are portable among hosts. The search engine supports phrase searching, relevance ranking, stemming, Boolean logic, and field searches.

The hard part about SWISH is the CGI interface. Many SWISH CGI implementations pipe the search query to the SWISH binary, capture the results, parse them, and return them accordingly. Recently a Perl as well as PHP modules have been developed allowing the developer to avoid this problem, but the modules are considered beta software.

Like Harvest, SWISH can "automagically" extract the content of HTML META tags and make this content field searchable. Assume you have a META tag in the header of your HTML document such as this:

```
<META NAME="subject" CONTENT="adaptive technologies; CIL (Computers In Libraries);">
```

The SWISH indexer would create a column in its underlying database named "subject" and insert into this column the values "adaptive technologies" and "CIL (Computers In Libraries)". You could then submit a query to SWISH such as this:

```
subject = "adaptive technologies"
```

This query would then find all the HTML documents in the index whose subject META tag contained this value resulting in a higher precision/recall ratio. This same technique works in Harvest as well, but since the results of a SWISH query are more easily malleable before they are returned to the Web browser, other things can be done with the SWISH results; SWISH results can easily be sorted by a specific field, or more importantly, SWISH results can be marked up before they are returned. For example, if your CGI interface supports the GET HTTP method, then the content of META tags can be marked up as hyperlinks allowing the user to easily address the perennial problem of "Find me more like this one."

WebGlimpse

WebGlimpse is a newer incarnation of the original Harvest software. Like Harvest, WebGlimpse relies on Glimpse to provide an indexing mechanism, but unlike Harvest, WebGlimpse does not provide a means to federate indexes through a broker. Compilation and installation is rather harmless, and the key to using this application effectively is the ability to edit a small configuration file that is used by the indexer (archive.cfg). Once edited correctly, another binary reads this file, crawls a local or remote file system, and indexes the content. The index(es) are then available through a simple CGI interface. Unfortunately, the output of the interface is not configurable unless the commercial version of the software is purchased. This is a real limitation, but on the other hand, the use of WebGlimpse does not require a separate pair of servers (a broker and/or a gatherer) running in order to operate. WebGlimpse reads Glimpse indexes directly.

Yaz/Zebra

The Yaz/Zebra combination is probably the best indexer/search engine solution for librarians who want to implement an open source Z39.50 interface. Z39.50 is an ANSI/NISO standard for information retrieval based on the idea of client/server computing before client/server computing was popularized:

It specifies procedures and structures for a client to search a database provided by a server, retrieve database records identified by a search, scan a term list, and sort a result set. Access control, resource control, extended services, and a "help" facility are also supported. The protocol addresses communication between corresponding information retrieval applications, the client and server (which may reside on different computers); it does not address interaction between the client and the end-user. --http://lcweb.loc.gov/z3950/agency/markup/01.html

Put another way, Z39.50 tries to facilitate a "query once, search many" interface to indexes in a truly standard way, and the Yaz/Zebra combination is probably the best open source solution to this problem.

Yaz is a toolkit allowing you to create Z39.50 clients and servers. Zebra is an indexer with a Z39.50 front-end. To make these tools work for you the first thing to be done is to download and compile the Yaz toolkit. Once installed you can feed documents to the Zebra indexer (it requires a few Yaz libraries) and make the documents available through the server. While the Yaz/Zebra combination does not come with a Perl API, you there are at least a couple of Perl modules available from CPAN providing Z39.50 interfaces. There is also a module called ZAP! (http://www.indexdata.dk/zap/) allowing you to embed a Z39.50 client into Apache.

There is absolutely nothing wrong with the Yaz/Zebra combination. It is well documented, standards-based, as well as easy to compile and install. The difficulty with this solution is the protocol, Z39.50. It is considered overly complicated and therefore the configuration files you must maintain and the formats of the files available for indexing are rather obtuse. If you require Z39.50, then this is the tool for you. If not, then something else might be

better suited to your needs.

Local examples

A number of local implementations of the various indexers reviewed here have been created. Use these links to play and see how well they work:

- freeWAIS-sf (plain text files where each "record" is delimited by a blank line)
- Harvest (plain text and HTML files across the Internet)
- Ht://Dig (HTML pages containing HTML META tags)
- Isite/Isearch (HTML pages containing HTML META tags)
- MPS (plain text files on the local file system)
- SWISH (HTML pages containing HTML META tags)
- WebGlimpse (HTML pages containing HTML META tags)

Summary and information systems

Indexers provide one means for "finding a needle in a haystack" but don't rely on it to satisfy people's information needs; information systems require well-structured data and consistently applied vocabularies in order to be truly useful.

Information systems can be defined as organized collections of information. In order to be accessed they require elements of readability, browsability, searchability, and finally interactive assistance. Readability is another word for usability. It connotes meaningful navigation, a sense of order, and a systematic layout. As the size of an information system increases it requires browsability -- an obvious organization of information that is usually embodied through the use of a controlled vocabulary. The browsable categories of Yahoo! are a good example. Searchability is necessary when a user seeks specific information and when the user can articulate their information need. Searchability flattens browsable collections. Finally, interactive assistance is necessary when an information system becomes very large or complex. Even though a particular piece of information exists in a system, it is quite likely a person will not find that information and may need help. Interactive assistance is that help mechanism.

By creating well-structured data you can supplement the searchability aspects of your information system. For example, if the data you have indexed is HTML, then insert META tags into your documents and use a controlled vocabulary -- a thesaurus -- to describe those documents. If you do this then you can use SWISH or Harvest to extract these tags and provide canned field searching access to your documents; freetext searches rely too much on statistical analysis and can not return as high precision/recall ratios as field searches. If your content is saved in a database, then it is an easy process to create your HTML and include META tags. Such a process is described in more detail in "Creating 'Smart' HTML pages with PHP" (http://www.infomotions.com/musings/smart-pages/).

The indexers reviewed here have different strengths and weaknesses. If your content is primarily HTML pages, then SWISH is most likely the application you would want to use. It is fast, easy to install, and since it comes with no user interface you can create your own with just about any scripting language.

If your content is not necessarily HTML files, but structured text files such database dumps, then MPS or the Yaz/Zebra combination may be something more of what you need. Both of these applications support a wide variety of file formats for indexing as well as the incorporation of standards.

Links

Here is a list of URL's pointing to the indexers reviewed in this text.

- freeWAIS-sf - http://ls6-www.informatik.uni-dortmund.de/ir/projects/freeWAIS-sf/
- Harvest - http://harvest.sourceforge.net/
- Ht://Dig - http://www.htdig.org/
- Isite/Isearch - http://www.etymon.com/Isearch/
- MPS - http://www.fsconsult.com/products/mps-server.html
- SWISH - http://sunsite.berkeley.edu/SWISH-E/
- WebGlimpse - http://webglimpse.net/
- Yaz/Zebra - http://indexdata.dk/zebra/

Chapter 5. Selected OSS

Introduction

Below is a list of open source software especially useful in libraries and open source software in general. This list is not intended to be comprehensive but selective instead. It is representative of the types of open source software available and the most used tools.

A more comprehensive lists of open source software especially designed for libraries can be found at OSS4Lib (http://www.oss4lib.org/). There you will also find the archives of the OSS4Lib mailing list, a low-traffic but ongoing discussion surrounding the issues of open source software in libraries. For an even more comprehensive list of software, check out SourceForge (http://sourceforge.net/). There you will find just about any type of open source software you desire.

Apache

Link: http://httpd.apache.org/

Apache is the most popular Web (HTTP) server on the Internet and a standard open source piece of software. It's name doesn't really have anything to do with American Indians. Instead, it's name comes from the way it is built. It is "a patchy" server, meaning that it is made up of many modular parts to create a coherent whole. This design philosophy has made the application very extensible. For example, there are the core modules that make up the server's ability to listen for connections, retrieve files, and return them to the requesting client (the "user agent" in HTTP parlance). There are other modules dealing with logging transactions and CGI (common gateway interface) scripting. Other modules allow you to rewrite incoming requests, manage email, implement the little-used HTTP PUT method, write other modules in Perl, or transform XML files using XSLT. Apache is currently at version 2.0, but for some reason many people are still using the 1.3 series. I don't really know why. I have not upgraded my Apache servers to version 2.0 because I do not want to loose the functionality of AxKit, an XML transformation engine. Apache is a part of LAMP (Linux Apache MySQL Perl/PHP), a term coined by RedHat to denote the core open source applications dealing with stuff Web.

CVS

Link: http://www.cvshome.org/

CVS is an acronym for Concurrent Versions System. It is the way open source software is shared by developers. It consists of a client and server application. The server is set up and points to a directory where one or more projects are saved. Usernames and passwords are created, and the server sits and waits for connections. For the most part, the CVS client is command-line driven. On the command-line you specify the location of a CVS server, the protocol you are going to use to connect to the server, and your username/password. Once logged in you give CVS various commands used to download remote projects. You then spend your time hacking away at the source code. When you think you have created the latest and great hack, you issue the CVS diff command to create a diff file. This file lists the changes you made to the original source. By sending this diff file to the project's maintainer, your hack can be incorporated into the next release. Alternatively, you might be granted write access to the remote project. In which case you issue CVS commit command, and your hacks are automatically incorporated. If you are going to do any open source software development, then you must get acquainted with CVS. Luckily, it comes pre-installed with many Unix variants, but it is just as easily compiled.

DocBook stylesheets

Link: http://docbook.sourceforge.net/projects/xsl/

Given a set of XML/DocBook files, the DocBook stylesheets, and/or an XSL processor such as xsltproc or FOP, you can transform your DocBook files into PDF documents, HTML documents, XHTML documents, or a few other file types. When you download the stylesheets, but sure to download the XSL sheets and not other types. You would need other processors to use the other types. The stylesheets are configurable by setting a number of parameters. Through this means you can specify a cascading stylesheet to be incorporated into your XHTML/HTML files. The stylesheets are thorough but do not allow you to change very much of the resulting output. If you don't like the way the stylesheets format your XML, you can always write your own stylesheets, but I'm willing to bet you have better things to do with your time. As a person who is interested in open source software, learning how to write DocBook files is a skill that will come in handy in the future.

FOP

Link: http://xml.apache.org/fop/

FOP is an implementation of the Formating Objects standards for transforming XML documents into documents intended for printing. It is mentioned here, not because it a primary open source software application, but because it is a Java application and represents a nice way to create PDF documents. For example, given an Java virtual machine, a DocBook file, the DocBook stylesheets, and FOP, you can create PDF versions of your DocBook documents. I have only had success with version 0.20.3 but it has proven indispensible a number of times. Writing FO stylesheets is not easy, and that is why I have relied on the DocBook FO stylesheets. Learning how to use FOP will give you good experience with Java as well as XML files.

GNU tools

Link: http://www.gnu.org/directory/

The GNU family of tools is wide and varied. Probably the most important one is gcc, a C compiler. Ironically, you can not compile the compiler unless you haves a compiler. Crazy. Consequently, beginning the process of software development is an sort of chicken and egg problem. For example, while you might be able to download the gcc distribution, but you will need gunzip and tar to uncompress the distribution, and you can't build gunzip nor tar without the compiler. No worry, many operating systems now come with an "unzipper" and a "de-tarrer". Frequently flavors of Unix (including Linux) come with a version of gcc pre-installed, allowing you to upgrade accordingly. Besides gcc, gunzip, and tar, there are a number of other very useful GNU tool including Berkeley DB (database library), binutils (miscellaneous binary utilities especially a linker and assembler), bison (alternative to yacc), curl (Internet user agent), emacs (text editor), fileutils (miscellaneous file utilities such as cp, mv, and rm), less (alternative to more), make (a sort of scripting language used to build source files), OpenSSH and OpenSSL (implementations of secure socket transactions), patch (applies diff files to source files), procmail (mail filter), sendmail (mail transfer agent), and wget (Internet user agent). By the way, and interesting discussion can be had by comparing the philosophy of "open source software" and GNU software.

Hypermail

Link: http://www.hypermail.org/

Hypermail converts email messages into sets of HTML files browsable by author, subject, date, thread, and attachment for the purpose of creating a mailing

list archive. As alluded to earlier, open source software is about communities. Email mailing lists are one of the primary, if not the primary, communication channels in the open source software world. As you develop open source software and manage a mailing list to keep everybody up-to-date, don't lets those valuable pieces of information go to Big Byte Heaven. Capture those "Perls" of wisdom by maintaining a mailing list archive with Hypermail. Hypermail is a C program driven by a number of configuration files and/or command line switches. Pass Hypermail raw, SMTP messages (Unix mbox files) and it will create sets fo browsable HTML files. The look, feel, and some functionality of the archives can be changed through templates and the configuration files. The only thing Hypermail does not support is searching the resulting archive. For that functionality you need an indexer, preferably an indexer that can index mbox files, but you usually end up using an indexer that can index HTML files.

Koha

Link: http://www.koha.org/

Koha is an integrated library system with a growing user community. Written in Perl and using MySQL as the underlying database, Koha makes it simple to create and manage a small integrated library system. Equipped with acquisitions, cataloging, circulation, and searching modules it provides much of the functionality of traditional online catalogs. With the recent implementation of its Z39.50 interface, it is easy to enter ISBN numbers into the system, locate MARC records, and have those records added. The user and system interfaces are simple and unencumbered, but alas, not very customizable. For many libraries, the catalog is the center piece of the operation. Koha represents a major step in providing a catalog that is functional and usable for small libraries. As long as support continues, I expect Koha to be more viable option for medium and possibly large library collections. The obstacle is not technology. The obstacle is time and effort.

MARC::Record

Link: http://marcpm.sourceforge.net/

This Perl module is the Perl module to use when reading and writing MARC records. It is very well supported on the Perl4Lib mailing list, and a testament to the module's abilities is its incorporation into things like Koha and Net::Z3950. If you are not familiar with object oriented programing techniques in Perl, then MARC::Record might take a bit of getting used to. On the other hand, learning to use MARC::Record will not only improve your programming abilities but it will educate you on the intricacies of the MARC record data structure, a structure that was designed in an era of scarce disk space, non-relational databases, and little or no network connectivity.

MyLibrary

Link: http://dewey.library.nd.edu/mylibrary/

MyLibrary is a user-driven, customizable interface to sets of library resources -- a portal. Technically, MyLibrary is a database-driven website application written in Perl. It requires a relational database application as foundation, and it currently supports MySQL and PostgreSQL. MyLibrary grew out of a number of focus group interviews where people said they were suffering from information overload. To address this problem, MyLibrary takes three essential components of librarianship (resources, patrons, and librarians) and tries to create relationships between them through the use of common controlled vocabularies such as a list of subject terms. Like a library catalog, MyLibrary provides the means to create collections of resources and classify these resources with a controlled vocabulary. Unlike a library catalog, the system also allows librarians as well as patrons to be classified in this man-

ner. By sharing a common set of controlled vocabulary terms relationships between resources, patrons, and librarians can be made thus addressing things like, "If you are like this, then these resources may be of interest", or "If you have this interest, then your librarian is...", or "These people have expressed an interest this, therefore your patrons are...", or potentially even doing Amazon-like things such as "People like you also used...".

MySQL

Link: http://www.mysql.com/

MySQL is a relational database application, pure and simple. Billed as "The World's Most Popular Open Source Database" MySQL certainly has a wide support in the Internet community. Many people think MySQL can't be very good because it is free, especially Oracle database administrators. True, it does not have all the features of Oracle, nor does it require a specially trained person to keep it up and running. A part of the LAMP suite, MySQL compiles easily on a multitude of platforms. It comes as a pre-compiled binary for Windows. It has been used to manage millions of records and gigabytes of data. Fast and robust, it supports the majority of people's relational database needs. On its down side, it does not currently support triggers, transactions, nor rollbacks. Nor does it have a GUI interface. At the same time, a program called phpMyAdmin, a set of PHP scripts, can be used to manage, manipulate, and query MySQL database through a Web browser window. If there were one technical skill I could teach the library profession, it would be the creating and maintenance of relational databases, and I would teach them how to use MySQL.

Perl

Link: http://www.perl.com/

Perl is a programming language. Originally written to handle various systems administration tasks, Perl's strength lies in its ability to manipulate strings (text). Perl matured through the era of Gopher but really started becoming popular with the advent to CGI scripting. Perl has been ported to just about any computer operating system, has one of the largest numbers of support forums, and has been written about in more books than you can count. Perl can be compiled into Apache making it possible to run Perl scripts as fast as C programs. It easily connects to database applications through a module called DBI. It can be run from the command line. It can listen and respond to networking connections. It can call many aspects of your computer's operating system. In short, Perl is mature and very robust. Other very good programming languages exist and can do much of what Perl can do. Examples include other "P" languages such as PHP and Python. These languages are becoming increasingly popular, especially PHP, but at the risk of starting a religious war, I advocate Perl because of its very large support base and its cross-platform functionality.

swish-e

Link: http://www.swish-e.org/

Swish-e is an uncomplicated indexer/search engine. Once built you feed the swish-e binary a configuration file and/or a set of command line switches to index content. This content can be individual files on a file system, files retrieved by crawling a website, or a stream of content from another application such as a database. The indexing half of swish-e is able to index specifically marked up text in XML and HTML as fields for searching later. The indexes created by swish-e are portable from file system to file system. The same binary that creates the indexes can be used to search the indexes. Swish-e supports relevance ranking, Boolean operations, right-hand truncation, field searching, and nested queries. Later versions of swish-e come with a C

and Perl API allowing developers to create CGI interfaces to these indexes. Swish-e is an unsung hero. It's inherently open nature allows for the creation of some very smart search engines supporting things like spelling correction, thesaurus intervention, and "best bets" implementations. Of all the different types of information services librarians provide, access to indexes is one of the biggest ones. With swish-e librarians could create their own indexes and rely on commercial bibliographic indexers less and less.

xsltproc

Link: http://xmlsoft.org/XSLT/

Xsltproc and its companion program, xmllint, are very useful applications for processing XML files with XSL. Both applications are built from a C library that is becoming increasingly popular for parsing and processing XML documents. By feeding xsltproc an XSL stylesheet and an XML data file you can transform the XML data file into any one of a number of text files whether they be SQL, (X)HTML, tab-delimited files, or even plain text files intended for printing. Xmllint is a syntax checker. Given an XML file, xmllint will check the validity of your XML files against a DTD. By first installing the C library and mod_perl, you will be able to incorporate AxKit into your Apache HTTP server allowing you to transform XML data on the fly and serve it accordingly. Swish-e desires the C library. It is easy to use the DocBook stylesheets with xsltproc to create XHTML versions of your DocBook files. With xsltproc and a plain o' text editor, you can learn a whole lot about XML.

YAZ and Zebra

Link: http://www.indexdata.dk/yaz/ and http://www.indexdata.dk/zebra/

YAZ is a C library and resulting binary application implementing a Z39.50/SRW client. Zebra is an indexer and Z39.50 server. The yaz-client is a straightforward terminal application. Zebraidx is the indexer, and requires bunches o' configuration files. It is not as straight-forward as other indexers, but its data can be served by zebrasrv. Since the client is built on a library, it can (and has) been compiled into other tools such as PHP and Apache. The YAZ API also has a Perl interface. YAZ/Zebra are definitely worth your time exploring if you want to make your collections available through Z39.50. Yes, you will spend time learning the in's and out's of Z39.50 in the process, but that experience can be taken forward and applied on other venues where Z39.50 is needed.

Chapter 6. Hands-on activities

Introduction

This part of the manual outlines the hands-on aspects of the workshop.

The activities outlined below were selected based on the software's popularity, the installation techniques they represent, the length of time and expertise they require, and their applicability to a library setting. This is not a comprehensive list of activities. A glaring omittion may be the installation of a number of GNU Tools, specifically, some sort of text editor, the compiler gcc, and make. Consequently, these activities assume the hosting (your) computer is duly equipped or the activities can be accomplished on top of Windows or Unix/Linux operating systems without compilation.

For the most part, the activities are listed in priority order; many times you must install a previous package before a subsequent package can be installed, but this is not always the case. All the packages to be installed in these exercises are included on the CD. Thus acquiring the software is a matter using the copy command (cp) from the CD to your home directory, or acquiring the software from the distribution site. The choice is yours.

The installation of open source and GNU software follows a pattern. You usually:

1. download the software
2. uncompressess and un-tar the package
3. run some sort of configuration program prior to compilation
4. compile (make) the software
5. test it
6. install it

Downloading the software is usually done through an FTP or HTTP interface. I like to get the URL of the remote file and feed it to a program called wget which then does all the work.

Uncompressing and un-tarring the is the work of gunzip and tar, respectively.

To configure for compilation there is usually some sort of file called configure or in the case of Perl modules you run the command "perl Makefile.PL". In either case the script examines the contents of your downloaded package to make sure it is complete, examines your computer's hardware and software to make sure you have the necessary tools installed, and finally builds some sort of a "make" file which is a script used to actually make the software. The most often used configuration is "--prefix". This configuration denotes where the software will eventually be installed. By default, most software gets installed in /usr/local. This is usually a good place, but circumstances are not always the same from person to person, so running a configuration like this, ./configure --prefix=/disk1/local, might be just what you need. When in doubt, try ./configure --help for a complete list configuration options.

In almost all cases the next step is to run make and the software is built. If there are problems, then you can usually run "make clean" to remove the mistakes, re-run the configuration script, and try make again.

Once the program is built, hopefully without errors, you might be able to run

"make test" which will examine whether or not program works.

Finally, you can run "make install" to put the program onto your file system. Access to /usr/local/bin, /usr/local/man, /usr/local/lib, /usr/local/etc, and /usr/local/include is usually restricted to root-level users. Consequently, you might need root privileges for this last step, but remember the --prefix configuration option. Using this option allows you to save the installation in your home directory. (Hint, hint!)

Installing and running Perl

In this exercise you will install Perl.

1. Acquire the Perl distribution by copying it from the CD, another location on your file system, or from the Internet. Save the distribution in your home directory.
2. Unzip the distribution with this command: gunzip perl-5.8.0.tar.gz.
3. Un-tar the distribution with this command: tar xvf perl-5.8.0.tar.
4. Change directories to the newly created distribution: cd perl-5.8.0.
5. Configure the build process with this command: ./Configure.
6. The configuration script will ask you lots o' questions. Accept the default answers to all of them except when it comes to where Perl and its supporting files will be installed. When asked these questions, specify your home directory like this: /home/[username] where [username] is your... username.
7. The configuration process takes a few minutes to complete, but when it is done simply run make by typing make at the command line.
8. The make process takes a number of minutes as well. Perl is very well written and will most likely make (compile) without any problems.
9. Test the make process using "make test".
10. Install the software by running "make install".
11. To verify that everything worked correctly, you should be able to type "perl -V" from the command line to see how things got built and where they got saved.

You can now run your first Perl script.

1. Copy the file named hello-world.pl from the extras directory of the CD to your home directory.
2. Examine the contents of the file with this command: more hello-world.pl.
3. Run the script like this: perl hello-world.pl
4. Open the script with pico, a text editor: pico hello-world.pl.
5. Change the contents of the print command.
6. Save your changes by pressing: ctrl-X

7. Go to Step #3 until satisfied.

Ta da! You have successfully installed Perl and run a Perl program.

Installing MySQL

Installing MySQL is the goal of this exercise. Be patient.

1. Acquire the distribution from the CD, Internet, or local file system.
2. Uncompress and untar it: gunzip mysql-4.0.13.tar.gz; tar xvf mysql-4.0.13.tar.
3. Change to the newly created directory: cd mysql-4.0.13.
4. Configure the installation and use many configuration options: ./configure --prefix=/home/[username]/mysql --with-unix-socket-path=/home/[username]/mysql/var/mysql.sock --with-tcp-port=[portnumber] --with-mysqld-user=[username] where [username] is your username and [portnumber] is a TCP port number assigned to you.
5. Compile the application: make.
6. Install the application: make install.
7. Initialize MySQL with this command: ./scripts/mysql_install_db
8. Give the root user of MySQL a password: /home/[username]/mysql/bin/mysqladmin -u root password [username] where [username] is your username.
9. Change directories to the location of your MySQL installation and start the server: cd ~/mysql; ./bin/mysqld_safe &. (To stop the server run: ~/mysql/bin/mysqladmin -uroot -p shutdown.)

In this exercise you will install some sample data into MySQL.

1. Make sure the MySQL server is running: ps -u[username] where [username] is your username.
2. Create a new database: mysqladmin -uroot -p create mylibrary. You will be prompted for a password, and use the password from the previous exercise.
3. Change directories to the extras directory of the CD and take a look some sample data: more mylibrary.sql.
4. Import the sample data into the new database: mysql -uroot -p mylibrary < mylibrary.sql.
5. Run the terminal-based MySQL client to begin to see the fruits of your labors: mysql -uroot -p mylibrary.
6. Once given the MySQL client prompt (mysql>), you can issue any of the following commands:

    - SELECT * from librarians;

- SELECT name, email_address FROM librarians ORDER BY name;
- EXPLAIN disciplines;
- SELECT discipline_name FROM disciplines;
- EXPLAIN items4librarians;
- SELECT name, discipline_name FROM librarians, items4librarians, disciplines WHERE librarians.librarian_id = items4librarians.librarian_id AND items4librarians.discipline_id = disciplines.discipline_id ORDER BY discipline_name;

Installing Apache

Here the basics of installing Apache are outlined.

1. Acquire the Apache software from the CD, local file system, or the Internet.
2. Unzip the distribution: gunzip apache_1.3.27.tar.gz.
3. Un-tar the archive: tar xvf apache_1.3.27.tar.
4. Change into the newly created directory: cd apache_1.3.27.
5. Run the configuration script making sure you specify your home directory as the prefix. In this case, it is also a good idea to put Apache in its own, separate directory like this: ./configure --with-port=[portnumber] where [portnumber] is the port number assigned to you - -prefix=/home/[username]/apache where [username] is your username
6. After the configuration files are created, make the server: make.
7. Finally, install: make install.
8. Should now be able to start the server: ~/apache/bin/apachectl start.
9. Verify that the server is working by connecting to it with your Web browser. The server's URL will be a combination of the IP address of your hosting computer and the value of Port described in Step #5, such as: http://127.0.0.1:8080/.

Now, create your own home page.

1. Copy the file named home.htm from the CD's extras directory to Apache's htdocs directory. The command will look something like this: cd home.html ~/apache/htdocs.
2. Take a look at the new file: more home.html.
3. View the home page in your Web browser with a URL looking something like this: http://127.0.0.1:8080/home.html.
4. Open home.html with your text editor: pico home.html.

5. Season (edit) it to taste and save the changes: ctrl-X.

6. Reload the home page: http://127.0.0.1:8080/home.html.

CVS

In this exercise you will install CVS.

1. Acquire the CVS "distro" from the Internet, CD, or local file system.

2. Use the usual technique for unpacking, making, and installing the application: gunzip cvs-1.12.1.tar.gz; tar xvf cvs-1.12.1.tar; cd cvs-1.12.1; ./configure --prefix=/home/[username] where [username] is your username; make; make install.

While the syntax is a bit confusing, retrieving a CVS repository is not too difficult.

1. Log into a repository, such as the one for MyLibrary: cvs -d :pserver:anonymous@dewey.library.nd.edu:/usr/local/cvsroot login

2. When prompted for a password, simply press return because the user anonymous does not have a password.

3. Download the repository: cvs -d :pserver:anonymous@dewey.library.nd.edu:/usr/local/cvsroot checkout MyLibrary.

4. In this exercise you will edit a file from the repository and create a "patch".

5. Use your favorite editor to change the contents of any text file in the repository: pico ChangeLog.

6. Save your changes: ctrl-X.

7. Create a "diff" file or patch: cvs diff -u ChangeLog > patch.txt

8. Take a look at the patch. It is the file you would send to the developer for inclusion into the repository: more patch.txt.

Hypermail

In this exercise you will create and install Hypermail. The process is pretty standard.

1. Download or copy the hypermail archive from the Internet or CD to your home directory: cp hypermail-2.1.8.tar.gz ~/.

2. Unzip the archive: gunzip hypermail-2.1.8.tar.gz.

3. Untar the archive: tar xvf hypermail-2.1.8.tar.

4. Change to the newly created directory: cd hypermail-2.1.8.

5. Configure the make process making sure to specify your home directory as

the prefix: ./configure --prefix=/home/[username] where [username] is your username.

6. Compile the program: make.
7. Install the program: make install.
8. You should now be able to run the program and read a simple help text: hypermail --help.

Do this exercise to create a browsable archive from a standard mail box file.

1. Make sure you have installed Apache, and make sure it is running.
2. Create a new directory under your Apache file system as a place to save your archive: mkdir ~/apache/htdocs/colldev.
3. From the extras directory of the CD, copy the supplied mail box file to the colldev directory: cp colldev.mbox ~/apache/htdocs/colldev.
4. Create a browsable index of the mail box with the following (long) command: hypermail -d ~/apache/htdocs/colldev -m ~/apache/htdocs/colldev/colldev.mbox -M.
5. Change directories to and list the items in the newly create directory. You should see bunches o' files as well as an index.html file: cd ~/apache/htdocs/colldev/; ls.
6. Finally, view the fruits of your labors in your Web browser: http://127.0.0.1:8080/colldev/.

If you have previously installed swish-e, then you can do this exercise where you will create a searchable index of your browsable archive.

1. Install swish-e.
2. Copy from the extras directory of the CD a swish-e configuration file to the colldev directory: cp swish-colldev.cfg ~/apache/htdocs/colldev.
3. Take a quickie look at the file. It contains extra instructions for the indexer (swish-e): more swish-colldev.cfg
4. Create an index: swishe-e -c ./swish-colldev.cfg -i / home/[username]/apache/htdocs/colldev/0*.html where [username] is your username.
5. You should now be able to search the index with simple swish-e commands: swish-e -w books.
6. Install a CGI script from the extras directory allowing you to search the index: cp swish-colldev.cgi ~/apache/cgi-bin.
7. Edit the script making sure its very first line points to your Perl binary: pico ~/apache/cgi-bin/swish-colldev.cgi. The very first line should look something like this: #!/home/[username]/bin/perl where [username] is your username.
8. Edit the line in the script that defines where the index resides. The line should read something like this: my $index = '/ home/[username]/apache/htdocs/colldv/index.swish-e'; where [username] is

your username.

9. Make the script executable: chmod +x swish-colldev.cgi.

10. 1Finally, give the script a whirl: http://127.0.0.1/cgi-bin/swish-colldev.cgi. Because Hypermail created structured data with meta tags, and because swish-e was configured to extract the meta tags and save them to specific fields, it is possible to do field searching against this email archive using queries like "subject = book".

MARC::Record

In this exercise you will install MARC::Record.

1. Obtain the MARC::Record distribution from the CD, local file system, or Internet.

2. Unzip it: gunzip MARC-Record-1.29.tar.gz.

3. Uncompress it: tar xvf MARC-Record-1.29.tar

4. Change into the newly created directory: cd MARC-Record-1.29.

5. Do the standard Perl installation procedure: perl Makefile.PL; make; make test; make install.

6. Take a look at the Perl documentation: perldoc MARC::Record.

I wish they were all this straight forward.

Next, you will use MARC::Record to extract author and title information from a set of MARC records.

1. Copy the files named marc-read.pl and marc-records.mrc from the extras directory of the CD to your home directory: cp marc-read.pl ~/ and cp marc-records.mrc ~/.

2. Take a peek at both of the files: more marc-read.pl and more marc-records.mrc. The first file is a simple Perl script to read author, title, and subject data from a file such as the second file.

3. Give the marc-read.pl script a whirl: perl ./marc-read.pl marc-records.mrc | more.

MARC::Record can also write MARC records. Here is an example demonstrating how:

1. Copy the files named marc-write.pl from the extras directory of the CD to your home directory: cp marc-write.pl ~/.

2. Take a look at the insides of the file: more marc-write.pl.

3. Run the script: perl marc-write.pl. (While doing your data-entry, you might have to press crtl-h to backspace and correct any mistakes you make.)

4. Go to Step #3 until you get tired.
5. Examine the fruits of your labors by feeding your marc-write.pl output file to marc-read.pl.

If you have installed YAZ, then you can do the following exercise to download MARC records from the Library of Congress.

1. Make sure you have installed the YAZ tool kit.
2. Install the Perl modules named Event and Net-Z395They are found on the CD. Both of these modules install in the normal Perl fashion: gunzip, untar, Perl Makefile.PL, make, make test, make install.
3. Copy the file named marc-get.pl from the extras directory of the CD to your home directory: cp marc-get.pl ~/.
4. Take a look at the file's insides: more marc-get.pl and notice how the remote database, server, and port are defined.
5. Equip yourself with a few ISBN numbers and feed them to marc-get.pl: perl marc-get.pl 0156005492 0812862279 0803272103 > catalog.mrc.
6. Browse your newly created catalog: perl marc-read.pl catalog.mrc.

swish-e

Use this process to install swish-e.

1. Acquire the swish-e distribution from the CD, file system, or Internet.
2. Uncompress and untar the distribution: gunzip swish-e-2.4.0-pr1.tar.gz; untar xvf swish-e-2.4.0-pr1.tar.
3. Change to the newly created directory: cd swish-e-2.4.0-pr1.
4. Configure the make process being sure to specify your home directory as the prefix: ./configure --prefix=/home/[username] where [username] is your username.
5. Compile, test, and install it: make; make test; make install.
6. Verify that things worked by running the newly created executable: ~/bin/swish-e -h. You should see a bunch o' command line switches that swish-e can use.

Now, let's index and search some data.

1. Copy the file named alawon.tar.gz from the extras directory to your home directory: cp alawon.tar.gz ~/.
2. Uncompress and untar the archive: guzip alawon.tar.gz; tar xvf alawon.tar.
3. Change to the newly created directory and examine any of the files using the more command. Each of the files is a little newsletter regularly put out by the American Library Association.

4. Make sure you are in the alawon directory and index the newsletter like this: swish-e -i *.txt.

5. Swish-e will output some diagnostic information. When it is complete list the contents of the alawon directory and notice the newly created files named index.swish-e and index.swish-e.prop. Combined, these files are your index.

6. Search the index with like this: swish-e -w [term] where [term] is a word or quoted phrase such as books or "library of congress". Swish-e should return a list of scores, file names, "titles", and sizes for each file that match your query.

While swish-e can be run from the command line, its real power is demonstrated though one of its programming interfaces. In this exercise you will install swish-e's Perl module and search the index with a Perl script.

1. Change into swish-e's distribution directory: cd ~/swish-e-2.4.0-pr1.

2. Change into the distributions's Perl directory: cd perl.

3. Use the standard Perl installation technique: Perl Makefile.PL; make; make test; make install. When complete you should be able to read the Perl documentation for swish-e: perldoc SWISH::API.

4. Return to the alawon directory: cd ~/alawon.

5. Copy the file named swish-alawon.pl from the extras directory of your CD to the alawon directory: cd swish-alawon.pl ~/alawon.

6. Take a look at the script: more swish-alawon.pl.

7. Run the script using queries you tried in the previous exercise: perl swish-alawon.pl. The resulting output should be a bit prettier.

YAZ

Here you will compile and install YAZ.

1. Acquire the "distro" from the CD, file system, or Internet.

2. Uncompress and untar the distribution: gunzip yaz-2.0.3.tar.gz; tar xvf yaz-2.0.3.tar.

3. Change directories to the newly created directory: cd yaz-2.0.3

4. Configure making sure to specify your home directory as the prefix: ./configure --prefix=/home/[username] where [username] is your username.

5. Make the application: make.

6. Install it: make install.

In this exercise you will search a Z39.50 target with the YAZ client.

1. Run the YAZ client: yaz-client.

2. Open a connection to the Library of Congress: open tcp:z3950.loc.gov:7090/voyager.

3. Do a simple free text search: f origami.

4. Display the first record: show 1.

5. Do a simple phrase search: f "structures of experience"

6. Show the first record: show 1

7. Do an ISBN search: f @attr 1=7 0156005492

Koha

In this exercise you will explore Koha.

1. Open your Web browser to the patron URL given to you in the workshop. Simply explore and play with the interface searching for items, reading detailed records, and creating an account for yourself.

2. Open your Web browser to the librarian URL given to you in the workshop. Notice the components that are available. Play with the librarian interface, specifically the acquisitions module, and try adding a few records via the Z39.50 interface or batch MARC records load process. (Remember, you might have a set of MARC records to play with from a previous exercise.)

3. Return to the patron interface and search for the items you added to the collection in Step #2.

MyLibrary

In this exercise you will explore MyLibrary.

1. Open your Web browser to the patron URL given to you in the workshop. Explore the interface noticing how the searching, browsing, and account creation/customization features operate.

2. In a second browser window, open the administrative interface with the URL given to you in the workshop.

3. Select the Global Message option from the Administrative interface, and use the resulting form to edit/submit the content of a global message.

4. Make the patron interface active by selecting your first browser window and reload the page. You should see the edits you made in the administrative interface.

5. Return to the administrative interface and use the Message from the Librarian option. Create edit/submit the content of the same discipline you chose when creating a MyLibrary account in Step #1.

6. Return to the patron interface, reload the page, and notice how the content of your page changes.

7. Return to the administrative interface and create a link to a new information resource by using the Reference Shelf, Databases, or Electronic Journals menu options.

8. Again, return to the patron interface, customize the content of your page, and notice how the resource you just added in the administrative interface is now an option in the patron interface.

9. Make the administrative interface active, and use the Create Static Pages option to create browsable lists of the information resources in the underlying MyLibrary database.

10. Make the patron interface active, and browse the newly created lists by using the All Resources link.

11. Make the administrative interface active, and use the Discipline Defaults menu option to create the defaults for a discipline of your choice.

12. Make the patron interface active. Log out. create a new account making sure you select the discipline you just modified, and notice how the defaults you created are manifested.

xsltproc

In this exercise you will install libxml2 and libxslt, the libraries necessary to run xsltproc. The process adheres pretty much to the standard GNU installation process: configure, make, make install.

1. Acquire the libxml2 library from the CD, local file system, or Internet and save it in your home directory.

2. Uncompress the distribution: gunzip libxml2-2.5.8.tar.gz.

3. Un-tar the distribution: tar xvf libxml2-2.5.8.tar

4. Change directories accordingly: cd libxml2-2.5.8.

5. Configure the build process remembering to specify your home directory as the prefix: ./configure --prefix=/home/[username] where [username] is your... username.

6. Build the library: make

7. Install the library: make install. When you are finished with this step there ought to be directory in your home directory named lib, and lib should contain a file named libxml2.

Now you will make a binary application that uses the libxml2 library, xsltproc.

1. Acquire the libxslt distribution from the CD, local file system, or the Internet and save it in your home directory.

2. Uncompress the distribution: gunzip libxslt-1.0.31.tar.gz.

3. Un-tar the distribution: tar xvf libxslt-1.0.31.tar.

4. Change into the newly created directory: cd libxslt-1.0.31.

5. Configure, making sure to specify your home directory as the prefix: ./configure --prefix=/home/[username] where [username] is your username.

6. Compile like this: make.

7. Install like this: make install.

8. When you are done you should have a binary named xsltproc in the bin directory of your home directory. You can run the command like this: xsltproc.

In this exercise you will transform an XML document into some other type of document using an XSL stylesheet and xsltproc.

1. Copy the files named hello-world.xml and hello-world.xsl from the extras directory of the CD to your home directory.

2. Take a look at the files like this: more hello-world.xml and more hello-world.xsl.

3. Do an XML transformation like this: xsltproc hello-world.xsl hello-world.xml.

4. Open hello-world.xml in a text editor: pico hello-world.xml.

5. Add a new message to the file and exit the editor by pressing ctrl-X.

6. Go to Step #3 until satisfied.

You can get a lot of use out of xsltproc, but the fact that it is distributed as a library than can be compiled into other applications make it even more powerful.

Chapter 7. GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice

placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

> a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
>
> b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
>
> c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

> a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by

third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License. 8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, IN-

CLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Discover what open source software is, how it works and get a detailed breakdown of its major advantages and disadvantages versus proprietary software.Â A complete guide to open source software, how it works and how it can be the free, flexible solution for any of your private or business needs. Published by Rachel Bridge, last update Jul 9, 2020. Open source software refers to software released without the usual copyright restrictions. This is a list of free and open-source software packages, computer software licensed under free software licenses and open-source licenses. Software that fits the Free Software Definition may be more appropriately called free software; the GNU project in particular objects to their works being referred to as open-source. For more information about the philosophical background for open-source software, see free software movement and Open Source Initiative. However, nearly all software meeting the Free Compare the best free open source Windows Library Software at SourceForge. Free, secure and fast Windows Library Software downloads from the largest Open Source applications and software directory.Â It stands upon the shoulders of many free/libre/open-source (FLOSS) libraries developed for processing low-resource languages, especially Persian and RTL languages Publications: Kashefi, O., Nasri, M., & Kanani, K. (2010). Towards Automatic Persian Spell Checking. SCICT.