Impediments to institutional adoption of Free/Open Source Software

Peter St. Onge
Information + Technology Services, University of Toronto
pete.stonge@utoronto.ca

Abstract

Free and Open Source Software (FOSS) have a number of characteristics that make it highly desirable in institutional settings: prevention of lock-in, cross-platform availability and version consistency across platforms, internationalization support, breadth and depth of choices, availability of updates and cost. Despite these manifold advantages, institutional uptake of FOSS has been limited. This paper discusses some of the key factors limiting adoption, and presents some suggestions on how these barriers can be overcome.

1 Introduction

This paper examines some of the strengths and weaknesses of FOSS as these relate to their use in institutional settings, environments where a large number of computers are in use – Typically this would involve anywhere from several tens of systems, upwards to several thousand systems in active central management.

Given that organizations are driven to save money and resources by realizing economies of scale wherever possible, the most important consideration in such a setting is the ability to manage three major components – users, systems, and services – as efficiently as possible. The user base is generally highly variable in terms of technical aptitudes, ability, and acceptance of any performance issues or anything impacting their ability to work, perceived or otherwise. The practical upshot of this approach is that any element requiring manual configuration or other "tweaking" is not suitable for use in an institutional context.

There are excellent examples of where FOSS-based systems are used as the foundation for institutional computing. This would include complex, multi-site file and print services for Windows client systems [30], institutional non-web single sign on [11], and corporate directories and authentication [3].

While extensive mention is made of Debian and derivative distributions in this paper, including the packaging mechanism, it is understood that similar facilities exist in RedHat/Fedora and other distributions of Linux and that the same points apply to these distributions as well.

2 Packaging and Distribution

Similar to commercial software development, the practices and processes underlying the development, packaging, and delivery of FOSS applications and systems can be highly variable even where best practices (eg. revision control, source code conventions, documentation conventions, etc) are relatively well known.

2.1 Distribution

While the distribution of software from both commercial and FOSS efforts can be done by the developers themselves, in the form of source tarballs or pre-compiled binaries, greater mindshare is typically achieved by making the application available via existing channel-based distribution systems, such as those used by Debian, RedHat, and derivatives. Beyond their philosophical differences underlying how FOSS-based systems should work (eg. Debian vs Ubuntu or RedHat or Fedora), all of the distributions of Linux serve the particularly crucial role of being the primary distribution channel for software in convenient form – packages. The importance of the distribution channel – the different Linux distributions – manifests in the fact that upstream developers often integrate distribution-specific tools to facilitate packaging the application (eg. in the Makefile); in this way, the amount of work required to effectively package an application for different distributions is minimized.

The distribution systems (eg. package repositories) as well as the packages themselves are difficult to subvert, thanks to the good use of md5 hashes for repository content lists themselves, for each of the many packages in that repository, and for files inside packages via manifest as required by the various distributions' policies[24]. It is thus possible to audit the integrity of system and application binaries on these FOSS-based systems, something is in theory possible but not mandatory when creating an MSI or EXE-based installation package. Newer, smaller organizations often do not have the expertise or resources to test their packages via build/install farm typical of larger organizations or projects[8].

The different "distros" each also provide quality assurance for those packages they make available. Distribution-specific policies set high standards for both source (eg. the dreaded FTBFS[13]) and binary packages[12]; packages are not permitted to enter into the distribution system until the policy requirements are met. Broadly-distributed tools such as linitian[25] and the like, in conjunction with the distributions' automatic build testing, help to ensure the ability to consistently install and uninstall individual packages cleanly and effectively.

In addition, these distribution mechanisms also allow for the provisioning of a package's dependencies (eg. shared libraries) in a separate package; in non-FOSS systems, these shared components were often shipped as part of a dependant package, often leading to conflicts in different versions of the same libraries installed on the system known as 'DLL Hell'[1], although this has become less of a problem in recent years[4].

Ultimately, these channels also provide a mechanism to provide patches back to the upstream source maintainers to ensure that underlying bugs can be resolved in a coordinated manner.

It is interesting to note that in the year 2010, the paradigm for distribution of software on the Windows operating system platforms is still primarily either the retail channel, or via downloads from the developers' sites, and that no central distribution channel exists. Moreover, no consist ant update mechanism has achieved any real traction: Microsoft's update mechanisms ("Windows Update" and "Microsoft Update") have focused primarily on their software with what appears to be limited participation by hardware vendors for driver updates (NVidia, ATI, Dell, Intel have occasional but infrequent updates), and other approaches like InstallShield's distribution service appear to have lost momentum after not gathering any real buy-in from software manufacturers.

Unfortunately, FOSS offerings for non-FOSS systems are also distributed in this same *ad hoc* manner for the most part, which acts to limit the knowledge of and access to the many multi-platform applications that would otherwise help build FOSS' mindshare outside of the existing areas.

This vacuum presents a tremendous opportunity for advocates of "World Domination" (as quoted in [7]) to put forward a packaging / distribution / update channel effort that would function effectively on non-FOSS platforms, typical of the home or unmanaged user. Being able to conveniently and easily install a FOSS package on a home user's non-FOSS system, in the way that **synaptic** would work, would be highly attractive to technical and non-technical users alike.

Not only would this approach help serve to bring FOSS to a much wider audience, it would also solve the problem of ensuring that end users can keep their applications up to date with minimal effort (eg. similar to how **update-notifier** already works).

Previous efforts have attempted to bridge this gap, strictly providing a distribution mechanism for packages: WPM[27] appears to have stalled during design phase, Win-Get[23] had more success as an active distribution channel, but the apps in channel appear to be quite dated (2.0.0.x for Firefox, Thunderbird). A third attempt, Appupdater[19] combines a distribution mechanism with a user application that tracks available and installed packages, and facilitates updates. It appears to be the most successful approach thus far, at least for unmanaged users in Windows, and continues to have a modest selection of updated software packages – 88 packages, some of which are FOSS (eg. Thunderbird, Firefox), some not (eg. Adobe PDF reader). This is rather a small number of applications, however, given that there are a rather large number of FOSS applications currently available for Windows.

2.2 Packaging

Even with a delivery mechanism to provide FOSS software to non-FOSS operating systems, however, there are other problems that remain particularly challenging for institutional adoption of FOSS.

From the perspective of the distribution, the main goal of a package is not just to facilitate the installation and removal of that package's application; it should also be to facilitate the future replacement of that package by another more updated version as it becomes available. Although it is possible to build software for distribution to non-FOSS platforms, it is generally not possible to build the distribution package for non-FOSS platforms (exe, msi) in the same way that is done for FOSS platforms (rpm, deb, etc).

Firefox, in particular, is a good example as a web browser popular with non-technical and technical people for the relative speed, flexibility, and general resistance to hostile sites[15]. Although there is a Windows binary installer package available, it cannot be used in managed Windows environments without first installing it to a test system, creating an MSI file based on the pre-install and post-install snapshots, then tested in a number of environments prior to being deployed via Group Policy Object (GPO)[33]. As the work required to package the application and test it properly is considerable, and given the frequency with which updates become available, it is no surprise that few administrators have opted to repackage applications themselves. This does create opportunities for enterprising individuals to provide such a service[17], and this has helped penetration of Firefox into some managed environments (eg. ours). Unless these are done as part of normal FOSS project activities, however, the project has no control over or voice in how the application is packaged and provided.

At present, there are multiple approaches used to package FOSS applications for non-FOSS operating systems. NSIS[6] is an open-source suite used to build executable (EXE) based installation packages; many commercial offerings exist as well (eg. InstallShield). These have all had a great deal of success in packaging FOSS applications for end users of Windows systems. From the perspective of managed systems, however, the standard packaging format for applications on Windows systems is the MSI[33]. Although other mechanisms for software installation over large numbers of centrally managed non-FOSS systems may exist, the best known and most used software delivery mechanism in managed environments typical of institutions is the MSI installer package installed automatically via GPO.

Ultimately, the ability to build MSI packages for FOSS applications for distribution on non-FOSS platforms, with the same care and attention as other packaging formats (deb, rpm, etc) to allow for package auditing and assurance, as well as the development of a distribution system similar to how FOSS software distribution already works, would remove a substantial barrier to awareness of the broad availability of FOSS on non-FOSS platforms in general, and to institutional adoption of FOSS on centrally-managed non-FOSS systems in particular.

3 Software

The advantages of FOSS in general have been treated extensively and exhaustively elsewhere[26, 34]. From the institutional perspective, there can be substantial wins in adopting FOSS: the cross-platform availability of given applications, support for internationalization, the tendency towards frequent and non-disruptive updates, and the ability to access support resources, both internal and external.

There are, however, substantial downsides that must be considered.

3.1 Cross-platform availability

Like numerous commercial applications, many established FOSS applications like OpenOffice or Firefox are available for multiple operating systems. Unlike many commercial applications that produce different versions of an application for different operating systems, however, FOSS applications tend to have consistent versions and interfaces across different operating systems.

The ability to have the same user experience from an application across multiple operating systems has important ramifications institutionally.

In terms of support, it becomes considerably simpler to diagnose and rectify user issues with the application, produce useful internal documentation to support the use of that application, and build institutional knowledge around the use of that applications.

From the user perspective, the application becomes the important element rather than the operating system. This can ease some pain points in heterogeneous computing environments. In particular, it allows users to be more "mobile" in terms of what operating system they use: A consistent interface across platforms minimizes user disorientation with the application, particularly when they already know that application well, even when the underlying operating system user interface differs from their "usual" platform.

This mobility offers an organization a great deal of flexibility in developing their IT strategy, as they can choose from different platforms based on their requirements while minimizing disruption from transitions.

3.2 Internationalization

In addition to the ability to have consistent versions of an application over multiple platforms, a further advantage of using established FOSS applications is well-known and well-supported ability for a single binary application to present its user interface elements in any number of languages[16].

Lacking such a well-documented practice, I have noticed that many commercial applications have separate releases to support individual languages. In many cases, this is effectively a different version release of a given application; the overhead of maintaining multiple concurrent versions of a commercial application makes integrating bug and feature fixes more complicated. The end result we witnessed was an incompatibility between an operating system of one language, and an application of another, despite the fact that both were from the same company. This has improved in Windows relatively recently[20] but still appears considerably more complex than how distros manage locales for i18n, and existing Mac OS X support.

In an organization spanning different areas of languages, dealing with language-specific software issues – particularly if a common vocabulary does not exist – adds considerable complexity to support and administration of these environments.

Conversely, a single binary application that has support for relevant languages is far more easily deployed and supported broadly, allowing for economies of scale and leveraging of administrative efforts required in an institutional context.

3.3 Frequent, small changes

The development cycle and distribution process of packaged binaries of FOSS applications provide a fairly painless means to integrate the relatively frequent and incremental updates over time. In the established or "stable" dists, security and bug fixes are unremarkable from the user standpoint as these minor internal issues and not major changes to user interface and other elements.

Even in cases where one decides to use the more dynamic "testing" or "unstable" dists in Debian, the experience of significant breakage is quite small in my own experience. This does not, of course, obviate the need to test packages prior to wide deployment.

This capacity for accommodating frequent minor changes provides the institutional administrator the means to take company-specific or resource-specific packages and maintain these in a state where minor changes be readily propagated in response to corporate, managerial or new policy requirements.

3.4 Support resources

In a large corporate organization, it is not unusual to have local staff whose primary role is to coordinate with a vendor for a particular product (or set of products), to manage pending bug or function issues, feature requests, and such. Although large software vendors often have well-developed self-help and similar documentation resources, dealing directly with support resources often requires the dedicated staff to handle contract and entitlement issues to ensure prompt access to updates and support resources.

After considerable work ensuring that entitlements and contracts are maintained, it was our experience as a relatively small organization (a large University) that days would usually pass between a support request and an initial response, we felt generally poorly served by such support arrangements. Especially in light of the cost of the support contracts for expensive software: Typically this was on the order of 20-25% of the original purchase price, with the percentage increasing on an annual basis.

As a result, serious consideration of alternative open source alternatives for some of these applications is being made at present[10]. Although in this case, justification to technical and managerial oversight was simple due to the organization's receptiveness to the use of FOSS, other organizations may be more hesitant for various reasons; perhaps the most important is the lack of formal "support" mechanisms.

As the number of organizations with similar problems opt for the FOSS solution, the number of active developers and participants involved with projects increases – each scratching their employers' itches as well as their own – these projects grow in capability and dependability. Having individuals in a position to support critical applications employed on-site, the responsiveness of support of FOSS in the form of immediate and direct access to bug fixes is a valuable asset. Ultimately, if critical vendor applications would require staff dedicated to support local customizations as well as coordinate with remote vendors, it is difficult to see how dedicating local staff to work on a FOSS project would imply any greater staffing costs. Furthermore, local staff involved in the development and support of internally developed applications can also be the source of considerable innovation for those organizations.

Greater visibility of corporate or institutional involvement in FOSS projects outside of the kernel, where this involvement is well known, would provide additional comfort to organizations considering adopting FOSS applications in important and visible areas.

3.5 Limitations

The breadth and depth of mature FOSS applications is a testament to the effectiveness of FOSS development methods. These applications, however, have a number of weaknesses that would have to be address to make them suitable for institutional use.

As one example, it is currently not possible for most of these very useful applications find all or part of their configuration elsewhere. User configuration (user name, email address, organization, incoming & outgoing mail server configurations) for a mail user agent (eg. Thunderbird), for instance, is still a usually a task for the user, or for a tech support representative. Web browser proxy configuration is a similar issue, particularly in corporate environments. Given the highly variable nature of user technical aptitudes in organizations, expecting users to enter in such information – simple as it may seem to many readers – can easily lead to unexpected overburdening of support resources and the loss of user acceptance of these applications.

In an environment where the mail service is managed centrally, and all users are known *a priori*, the inability to provision user data to the application on behalf of the user reflects poorly on the technology and those who administer it.

The ultimate goal of such efforts is to limit the need for manual configuration of production user applications – be they mail clients, web browsers, database clients, databases, ODBC connections, etc. This not only allows for a more productive user experience (minimizing time lost due to configuration issues), it also provides a means to minimize impacts of future service changes (different server software, different IP address used, etc).

4 Operating Systems

FOSS-based operating systems have had good support via PAM for external identity management systems for some time now, Kerberos and LDAP have been two of the most commonly used of FOSS-based systems[28, 11] for that purpose.

The primary goal of external authentication frameworks like these is twofold. First, to shrink user credential space by reducing the number of credentials that users in an organization have to use to prove their identity to non-critical services on a daily basis. Making it easier for the user to remember their user name and one good password (which can be enforced via password policy, of course) rather than requiring many user names and corresponding (and usually bad) passwords on different systems – usually on systems where password policies cannot be implemented – can drastically reduce the number of password reset requests and thus reduce demands on IT help desks. Moreover, the common user name across multiple systems allows for more effective auditing of user activities across these systems, simplifying the detection of anomalous behaviour (eg. access to systems from "local" IP addresses concurrently with access via VPN from "foreign" IP addresses).

The second, and perhaps more important goal, is to provide means for identity to be vouched for by a separate trusted system (eg. Kerberos) once the user successfully authenticates themselves; this process is often referred to as Single Sign On[32], or SSO. From the user's perspective, SSO reduces issues and inconveniences around identification and authentication by having these handled transparently after the initial authentication. Log in once, and never again for that day.

From the institutional perspective, however, SSO provides another important value. SSO reduces password exposure by drastically reducing the number of information system that have to handle passwords and hence reducing the avenues for user credential compromise. In other words, by relying on the Kerberos (or similar) service for authentication, servers/services do not ever handle the user's password, and their compromise is less likely to facilitate exploiting other systems via captured user credentials.

Moreover, SSO simplifies services to some extent by obviating the need to design and implement user credential (user/pass) facilities, allowing developers to re-use code dealing with the underlying SSO service for those purposes.

Single sign on does not obviate the need for additional levels of protection and authentication around critical institutional resources; services such as payroll, finance, etc. require greater protection than the individual user's machine, for instance, and the use of two-factor (or more) authentication is called for in these circumstances.

Kerberos is perhaps the most mature SSO systems, and remains important, particularly because it remains the only SSO solution that bridges both system-level authentication and web single sign on through web browsers[21].

Given the competing paradigms for identity management in play (Active Directory[5], LDAP[2], Kerberos[11], PubCookie[22], Shibboleth[14], and others), it would appear that institutional systems will need to accommodate more than one of these approaches in heterogeneous institutional environments.

5 Practice

Although the technological capacity for FOSS-based systems to support institutional is already well-established through the use of directory services[31], the understanding of what directory services can provide is still very limited by most administrators of FOSS systems.

In discussions with colleagues both near and far involved in managing large numbers of machines in different environments, and generally the adoption of directory services to support administration is minimal. As a result, the community of practice around the use of directories to support FOSS-based or more heterogeneous environments remains relatively underdeveloped.

5.1 Directory services

The Lightweight Directory Access Protocol[35, 29] (LDAP), is a directory service allowing for user, group, machine, authorization, and service information to be maintained centrally and in a secure manner. OpenLDAP provides a standards-based implementation of LDAP[3], and Active Directory extends LDAP to support Microsoft Windows-based systems and services[5].

Central control of any resource usually invokes political and other concerns. One of the advantages of directory services is the ability to delegate control over parts of the directory tree to particular individuals and groups, allowing local control over access and facilitating managerial tasks through a common GUI or web-based interface.

The adoption of LDAP and AD by FOSS applications manifests in many different areas. What follows is not exhaustive by any means, but does serve to show possibilities.

Mail user agents (Thunderbird and Evolution are only two examples among others), for instance, can access corporate directory information from an LDAP directory. The PostgreSQL database can have its service list in stored in LDAP, which facilitates connecting to remote institutional database servers.

Many office appliances (scanners, copiers, faxes) can make use of user information from an LDAP server.

The ISC DHCP and BIND servers can both use LDAP to contain their respective data; not only can this allow for the management of data across multiple DNS / DHCP servers across the institutional network, but they also provide a means to ensure service redundancy inside the institution.

Samba can use directories to manage user, group, and machine elements in large organizations through the use of LDAP to contain user account information[30].

PAM can also make use of user and group info in LDAP, and with the proper components, new user home directories can be created at the initial login. Further, the automounter can use information in LDAP to mount the user's remote home directory via NFS3 or NFS4.

The Puppet datacenter automation tool can make use of LDAP to store configuration info for the machines that it controls[18].

5.2 What's missing...

Even with delegation and the breadth of applications able to make use of directory services, there are a number of application shortcomings that remain.

As mentioned previously, the ability to tell user applications where to look for their configurations (eg. having the user's email application find the user's info in a particular directory location) as a means to auto-provision applications is an obvious institutional example.

For applications or services serving multiple users or systems, however, similar benefits could accrue. Storing application configurations in a directory, where appropriate, would allow for the ability to check and modify the application's configuration remotely and non-intrusively.

The benefits of directories are not always well-known, particularly since these usually only become used in larger environments not commonly experienced by a sizable majority of FOSS users and developers.

The perception that the cost of one-off changes to systems to allow them to keep working or participate in the local environment is far smaller than the benefit that a well-designed directory service can provide; in aggregate, however, the reverse is true. Not only would directory services provide am effective way to manage users and equipment, but as mentioned above, would allow user data to be provisioned appropriately across platforms.

Another barrier in the FOSS world, the directory server tree has no default population at install time. The nature of most system administrators is to take the precautionary approach in that they would want to understand how a system works prior to configuring it. Combined with the highly flexible nature of directories, in my experience the ethereal nature of a directory makes it harder for most to grasp. In addition, configuring individual machines or services to use the directory is a similarly involved task, at least initially.

Unlike FOSS directory servers like OpenLDAP, Active Directory-based directory servers come pre-populated to accommodate the majority of common tasks (eg. management of users, groups, and machines), primarily through information gathered when the server is initially installed. The process of enrolling Windows machines is also relatively straightforward at install time.

In order to realize the manifold benefits of directory services supporting the use of FOSS in organizations, a greater community of practice around the use of these techniques is needed. As this becomes more established, other shortcomings at the level of software are more likely to be addressed[9]. As in other communities around technologies (eg. Samba), an intrinsic activity this community should undertake is to document examples where directory services are used to support FOSS systems, the problems encountered and lessons learned (good and bad) in the process of set up and maintenance of the directory, as well as best practices to ensure continuity.

6 Conclusions

Presently, the underlying technology required to productively support the use of FOSS-based systems at the institutional level exists at both the software application and operating system levels. The majority of the technical limitations remaining involve the ability to for applications to obtain user-specific information through directory services.

It is expected that as FOSS applications are increasingly adopted by institutional users that support for packaging FOSS applications for non-FOSS platforms will become more commonplace, akin to how FOSS applications are packaged for FOSS-based platforms, although this would likely require a FOSS means to create MSI files.

Finally, the most important limit to the potential of directory services is the general lack of knowledge of how directory services work and how to properly design directories for particular situations.

7 Acknowledgements

This paper benefited immensely from discussions with a number of people, and I would like to thank them: Ian

Thomas, Martin Loeffler, Mike Wiseman, David Auclair, Peter Eden, John DiMarco, Ted Sikorski, David Sutherland, Roger Dingledine, and Richard Sanford.

Any inaccuracies or errors are but my own.

References

[1] Rick Anderson. The End of DLL Hell. `http://msdn.microsoft.com/en-us/library/ms811694.aspx`.

[2] Brian Arkills. *LDAP Directories Explained: An Introduction and Analysis*. Addison-Wesley Professional, 2003.

[3] Gerald Carter. *LDAP System Administration*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.

[4] Raymond Chen. Windows Confidential: Getting Out of DLL Hell. `http://technet.microsoft.com/en-ca/magazine/2007.01.windowsconfidential%.aspx`.

[5] Brian Desmond, Joe Richards, Robbie Allen, and Alistair Lowe-Norris. *Active Directory: Designing, Deploying, and Running Active Directory*. O'Reilly Media, Inc., 2008.

[6] The NSIS developers. The Nullsoft Scriptable Install System (NSIS). `http://nsis.sourceforge.net/Main_Page`.

[7] Chris DiBona, Sam Ockham, and Mark Stone. *Open Sources: Voices from the Open Source Revolution*. O'Reilly Media, 1999.

[8] Andrew Dunstan. PostgreSQL BuildFarm. `http://buildfarm.postgresql.org/`.

[9] The Apache Software Foundation. Apache Directory Project. `http://directory.apache.org/`.

[10] The Kuali Foundation. About the Kuali Community. `http://www.kuali.org/about`.

[11] Jason Garman. *Kerberos: The Definitive Guide*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.

[12] The Debian QA Group. Debian quality assurance. `http://qa.debian.org/`.

[13] The Debian QA Group. Debian wiki: FTBFS. `http://wiki.debian.org/qa.debian.org/FTBFS`.

[14] Internet2. Shibboleth. `http://shibboleth.internet2.edu/`.

[15] Brian Krebs. A Peek Inside the 'Eleonore' Browser Exploit kit. `http://krebsonsecurity.com/2010/01/a-peek-inside-the-eleonore-browser-e%xploit-kit/`.

[16] Tomohiro Kubota. Introduction to i18n. `http://www.debian.org/doc/manuals/intro-i18n/`.

[17] Ing-Long Eric Kuo. Firefox MSI. `http://www.frontmotion.com/Firefox/`.

[18] Puppet Labs. Storing Node Information in LDAP. `http://projects.puppetlabs.com/projects/puppet/wiki/Ldap_Nodes`.

[19] Neil McNab. Appupdater. `http://www.nabber.org/projects/appupdater/`.

[20] Microsoft. Guide to Windows Vista Multilingual User Interface. `http://technet.microsoft.com/en-us/library/cc721887(WS.10).aspx`.

[21] University of Maryland Office of Information Technology. Configuring Web Browsers for Kerberos Authentication. `http://www.helpdesk.umd.edu/topics/applications/kerberos/4782/`.

[22] University of Washington Techology Services. Pubcookie: open-source software for intra-institutional web authentication. `http://www.pubcookie.org/`.

[23] Ryan Proctor. Win-get. `http://windows-get.sourceforge.net/`.

[24] The Debian Project. Debian Policy Manual. `http://www.debian.org/doc/debian-policy/`.

[25] The Debian Project. Lintian. `http://lintian.debian.org/`.

[26] Eric S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001. Foreword By-Young, Bob.

[27] Edward Ropple. WPM - A Windows Package Manager. `http://blacken.superbusnet.com/oss/wpm/`.

[28] Andrew Ryan. HOWTO-pam. `https://mon.wiki.kernel.org/index.php/HOWTO-pam`.

[29] J. Sermersheim. RFC4511: Lightweight Directory Access Protocol (LDAP): The Protocol. `http://tools.ietf.org/html/rfc4511`.

[30] John H. Terpstra. *Samba-3 by Example: Practical Exercises to Successful Deployment (Bruce Perens Open Source)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[31] Wikipedia. Directory Sservices. `http://en.wikipedia.org/wiki/Directory_service`.

[32] Wikipedia. Single sign-on. `http://en.wikipedia.org/wiki/Single_sign-on`.

[33] Wikipedia. Windows Installer. `http://en.wikipedia.org/wiki/Windows_Installer`.

[34] Sam Williams. *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[35] K. Zeilenga. RFC4510 Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map. `http://tools.ietf.org/html/rfc4510`.

Proceedings of the Linux Symposium

July 13th–16th, 2010
Ottawa, Ontario
Canada

Conference Organizers

Programme Committee

Proceedings Committee