10 Reasons Why I Prefer Linux Over Windows

John S. Riley, Ph.D.

28th June 2005

Contents

Introduction

The OS debate rages as some take sides with nearly militant fanaticism. Here I present the sides from my perspective: a more-than-casual user who relies on computers as professional tools. As always, "your mileage may vary;" however, I have tried to present this list as issues one might wish to consider when choosing an OS for their computer.

The bottom line is, of course, choice. There is no product or tool that is all things for all people; for some uses, a ball peen hammer is better whereas other tasks are better suited to a claw hammer. It is therefore intriguing to me that so many IT types envision conversion or market dominance as the goal. With certain qualifications, I really don't care if another user or business owner prefers Windows; I just know that for me, Linux is a far superior product.

Before presenting my little list, I'd like to address one other point. Newcomers to the 'debate' will often see geek-speak phrases like "Linux is not ready for prime time, *yet*," or "it's just not ready for the desktop," etc. As evidence for these types of comments, specific anecdotes are offered as to why Linux has flaws that render it unusable for a particular case; the problem is that these anecdotes (often not technically understood by the speaker) are extrapolated to *all cases*. This is pure rubbish. In this article, I present my own anecdotes that contradict this 'conventional wisdom;' I do not ask that my experiences be extrapolated to all cases. Take these comments as you wish.

There are many millions of users worldwide using Linux. These users range from small children to technology professionals; use environments range from home Internet surfers to very complex wide area corporate networks. Most modern hardware is supported, and a lot that is not is cheap junk anyway. So, if we are going to debate the merits of operating systems, let us at least be intellectually honest. Both Windows and Linux are capable of running modern computers in a variety of settings for a variety of users, and the debate should begin with both on equal footing.

1 Linux Is Open Source

Since the 1980's, before Linux, groups of Unix developers began to believe that software should be free and freely used. This idea gained considerable steam in the 1990's when Linux became the flag-ship project of the so-called Open Source movement. Open Source Software (OSS) is a 'new' software development paradigm that is in direct contrast to the traditional, proprietary development method. But these development styles differ in many ways, not just in whether the source code is available or not.

In the proprietary model, development occurs for years and the released product is generally very polished. When bugs are found or new features requested by users, it may be years before a new release incorporates changes. This is the 'release late-release rarely' development model. Intrinsic to this model is that software is a *product*, packaged for consumption and purchased by users as a 'finished product.' Key here is the role users play: consumers. One problem with this idea is that users cannot ever be completely removed from the development process, but their role in the proprietary development model is rather limited (in most cases).

Many OSS projects are done on a 'release early-release often' model. This means that the developers release software *as it's being developed.* The software is not viewed as a product, but a means to an end, such as an increase in productivity. Users get snapshots of applications and tools *before* they are complete (feature-wise) or polished, and changes are incorporated rather quickly. Users can input ideas, feature requests or report bugs. But even more importantly, users can modify the programs themselves to suit their needs if the development is not to their liking. This is key, and lies at the basis of the term 'Free' in Free Software as it is applied in the Open Source Movement.

Most OSS is 'free,' but the use of this term causes considerable confusion. Many believe that free means "free as in lunch." This is only part of the story. While it is true that a large quantity of Open Source Software can be had for no money, the real meaning of free is "free as in speech." That is, when a user uses free software, he is free to use it as he wishes. He can change it, give it away to his friends, or even, in most cases, sell it for profit. This is fundamentally different than most proprietary software for which one does not even own the product one purchased (but rather a license to use it under certain conditions).

A manager considering the appropriate platform to meet his business needs should consider this carefully. MS Windows is the archetypical 'software product,' complete with product lifecycles, end of support and a 'closed' format that means MS controls the development. Linux, being OSS, can be supported indefinately in-house if need be by in-house staff; one can obtain Linux for free, you never have to pay for an upgrade and Linux will never see "end of life" so long as one chooses to maintain it. Indeed, many OSS projects, especially big ones like the Linux kernel, have such large a user and developer base that even if the current maintainers dropped their projects, others would likely take over.

Advocates of proprietary software, such as Windows, like to posit that documentation is better than for OSS. This depends on the software being considered. I've seen really terrible documentation for both proprietary and OSS software, and I've seen excellent documentation for both as well. Linux itself is extremely well documented. To me, the documentation for Linux makes more sense, since with Windows, MS seems to redefine every term in computing to their own. Reading MS Windows documentation is like reading something written and proofread by insiders, marketing and legal corporate bureaucrats. To me, personally, the documentation for Linux and related software appears to be written by real people trying to solve the real problem they are addressing.

Finally, being Open Source, Linux is infinitely configurable. You can change *anything* about the Operating System you would like; you cannot do that with Windows; you have to take what is given to you.

2 Linux Grows with Your Needs

One of the most frustrating things I have encountered in assessing the IT needs of my company is that since I cannot always predict what I will need *next year*, when next year arrives, I lack the infrastructure to accomplish my tasks. When I first starting running Windows 2000 on a main office computer, I thought 'gee this is pretty good, and much better than Windows 98.' However, it was only a matter of weeks before I ran into a situation that what I needed was really Windows 2000 Server; I certainly did not have the money for that, so I did without. This happened repeatedly over the course of several years, and never became a fully acceptable way to operate.

I have a colleague that serves as a Windows administrator for an organization; he does not make purchasing decisions but can only 'advise' those that do. The network was originally Windows XP Home Edition with one Windows XP

Professional computer functioning as a file server. The system has many, many problems, partly due to weak enforcement of use policies and non-centralized user authentication. The problem is due to Windows XP Home not being able to join a domain with the domain controller providing access control to the network. All admin involves sneaker net deployment to the individual clients and much redundancy in human-computer interaction.

The organization then bought a software package that required all users to essentially be simultaneously connected to the file server. Since the network size exceeded the 10 user max in Windows XP Pro, productivity crashed. The decision was made to purchase (for about $1000) Windows 2003 Server to handle the simultaneous connection issue. Existing software did not work on Windows 2003 Server, so the Windows XP Pro box was kept to serve data for those applications. The clients are still XP Home, so a domain controller is still not possible. Clearly, the licensing cost of upgrading to XP Pro for the clients is prohibiting proper and efficient network implementation.

In contrast, Linux operates within a model that can be summarized as "get things done." It is a workhorse system, not a manipulated means to a revenue stream. Last year, I needed to use an application that communicated in parallel with other nodes on the network using Kerberos. In a matter of hours, I installed and configured the Kerberos server(s) and the clients. No additional software purchase was necessary. Similar cases arise nearly weekly in my business, and I've yet to have to purchase a higher class license or additional software for getting things done with my Linux based systems.

3 Linux Was Multi-User by Design

Around 1985 or so, the IBM PC running MS-DOS was becoming popular both as personal and business computing platforms. The 8086 processor was real mode only and was designed only for single processing, single use. Though the 80286, then the 80386, could handle protected mode operation at the hardware level, the Operating System these systems ran was generally still MS-DOS running in real mode. Protected mode systems were available at that time, but were not in common use. The advent of MS Windows brought basic multitasking and protected mode to the PC, but the basic Windows design remained single user. Typical home user Windows systems did not become truly multi-user capable until the release of Windows 2000.

Though almost all current releases of Windows are multi-user, Windows seems like it has difficulty shaking its single user roots. Some versions of Windows have limits on the number of simultaneous connections or sessions that are supported.

Linux on the other hand was multi-user from the start. Linux, which began around 1991, was an evolution of an Operating System called Minics which in turn was written to teach Operating System Design. The core functionality loosely followed that of Unix, which was 20 years mature at that time. This is key to understand some of the core differences between Linux and Windows:

Linux utilized the ideas of a multi-user, heavily networked, strongly tested OS with a 20 year history.

4 The Linux Software Deployment Model Favors Performance

When you install any version of Windows on a computer, one conspicuous software tool is missing: a compiler. The software deployment model for Windows based systems is solely deployment of executable programs. In addition to the flexibility considerations mentioned in Section 1: Linux Is Open Source, this implies that software is 'compatible' with the lowest common denominator for the target audience of the software. This lowest common denominator could be Windows version, CPU type, memory configuration, video/graphics capability, etc. For example, a user running an Athlon XP or Pentium 4 processor is generally forced to use software compiled for a Pentium (or maybe Pentium II). This in turn implies the user cannot upgrade to new and improved libraries and re-compile for better performance (one MS solution to this particular issue is the COM model such as used with DirectX; while reasonable for a graphics library, no one would argue this is extremely complicated approach for simple programs).

To provide a point of contrast, consider the installation of software one obtains from the Internet. To install software obtained from the Internet, one would perform the following steps:

1. Download the binary package; usually many files in a single, compressed distribution file.

2. If required, uncompress the package into a working directory. This requires the proper uncompression utility, which is not typically included with Windows.

3. In the directory into which the package was uncompressed, double click the file setup.exe.

4. Respond to any input the package installation program requires. This can be easy or complicated, depending on the package.

With this binary-only deployment model, the software vendors choose the 'minimum system requirements' and compile for that target. More advanced systems get little advantage. For an overblown example, imagine you need a program to do a task, but it was written to run on a 80486 processor. While your new Pentium 4 *will* run the software more efficiently, a large number of the new, highly advanced features of the Pentium 4 are not used. Couple this with the fact that the user cannot customize the software for a particular use (because the source code is not included and Windows does not ship with a compiler) and one is forced to conclude binary-only deployment is not very flexible.

Linux, on the other hand, generally ships and gets installed with a compiler. Many software packages for Linux are therefore available as binary packages or as source code. Users wanting basic features and 'one-click' installation get it; those wishing more flexibility, or higher performance, can compile the code themselves, *optimized for their machines*. It should be pointed out that software installation on Linux systems is one of those cases many people often criticize; personally, I find package installation as easy as that on Windows. Command line tools such as rpm or apt-get are straightforward, but for those wishing GUI tools, rpmdrake and KPackage are two examples that work well. Further, with KDE and Konqueror, 'one-click' on a .rpm file will begin the installation process much like clicking a setup.exe file in Windows. Installation of binary software using rpm on Linux can be done using the following steps (compare to those for Windows, 4):

1. Download the binary package; usually many files in a single, compressed distribution file.
2. In the directory into which the package was downloaded, double click the .rpm file.
3. Respond to any input the package installation program requires. This can be easy or complicated, depending on the package.

Clearly, this is not any more complicated than installing software on Windows.

I should also emphasize that compiling from source is not hard in many instances. I find myself often preferring to install from source since a generally better performing program results. Also, custom compilation allows the user to compile with certain features enabled or disabled; if there is a feature of a specific application that you know you will never use, you can compile without it, thus creating a smaller and possibly faster application. With modern compilation tools, compiling from source is typically characterized by the following steps:

1. Download the source code, which is usually many files in a single, compressed file for distribution.
2. If required, uncompress the package into a working directory. This requires the proper uncompression utility, which is typically included with Linux.
3. In the source directory (where the files were written after uncompression), read the README and INSTALL files for specific notes and requirements.
4. Using the Command Line Interface (sometimes called a command prompt Windows-speak), issue the command: ./configure
5. Again at the Command Line Interface, issue the command: make
6. At the Command Line Interface, become the superuser: su
7. At the Command Line Interface, issue the command: make install

Steps 1-2 are essentially the same as for binary distribution on Windows. While compilation may seem more complicated, it is generally quite easy. You download, uncompress, check for specifics in README and INSTALL, do ./configure, make, make install (as root). Compilation itself can take a while (a few seconds to hours, depending on the package and the speed of the computer), so there is a trade-off to the higher performance, more flexible option.

A key step here is Step 4, configuration. At this step, the Makefile is generated that contains commands for the compiler. Here, one can fine-tune the compilation process, specifying compiler optimization flags, compile in features (or disable features you know you will never need, which is also useful), etc. Some details of your system are probed and compiler settings adjusted accordingly. The cost of compiling your own, custom version of the software you download is relatively small and is far outweighed by the benefits you gain.

In summary, with software for Windows, you get what you get. Software vendors cannot assume that you have a compiler (if they want to supply source code, which is often not the case anyway), so 'must' supply an executable program. So that they do not have to maintain many, many copies of the same program, they compile for the lowest end system in their target market. Linux, on the other hand, provides the user with the option to use pre-compiled binaries or to compile from source. This is much more powerful and fits the needs of a far broader spectrum of users.

5 Security Is Not an Afterthought in the Linux Design

One of the single biggest reasons many users migrate from Windows to Linux is increased security. So, what is it about Linux that makes it more secure? The answer is simple: design. Modern Windows (XP) is the marriage of two development trees of the Windows Operating System: the 9x and NT trees. Windows NT was the professional version that grew from the VMS operating system; The 9x tree, includes Windows 3.1, 95, 98 and ME. It is these versions of Windows that became popular with users, but it is the also the ties to the 9x tree that is the security problem.

The Windows 9x tree is *single user* and went much futher than that in terms of lack of security. Since they were single user, the user was by definition 'administrator' and had full access to all system resources. Exploited vulnerabities therefore could not be contained to a non-priviledged userspace. Usernames were only used for network identification, not really as a true authentication mechanism. In addition, these systems were based on file systems (FAT16 and FAT32) that did not have file permissions.

Even in the NT tree, the situation is not much better. You can define priveledged and non-priveledged users in NT based systems, but there many flaws in the design. A lot of software, including some written by MS itself, will not run properly unless run by a priveledged user. In addition, there is

no easy way to gain priveledge for certain tasks. These two facts conspire to push users to run as privileged users thus negating the supposed 'security' in using limited user accounts. Beyond user privilege, while the NTFS file system does provide file permissions, the default settings are wide open and one must manually restrict access where desired. Further, NT based systems have the "Everyone" user which must be handled very carefully to secure a Windows system.

As stated, Windows XP, formally Windows NT 5.1 but with 'features' to appeal to home users used to the 9x products, is a marriage of these two Windows trees. Windows XP Home has only rudimentary file permissions, the limited user accounts are too restrictive (many applications do not work properly) and cannot join a network domain. Windows XP Professional is a typical NT tree product.

Linux, a derivative of the Unix OS, was multi-user from its inception. The concept of privilege was built into the design from the start and rests on the maturity (two decades of real world testing) of Unix. Except for system tools, most applications in Linux do run properly when run as an 'ordinary' user; to gain privilege, one can "su" and provide the administrative (root) password. File systems provide per-file permissions for 'user,' 'group' and 'other.' It is easy to configure Linux so that default file permissions are quite restrictive, forcing the administrator to willfully 'open' the security where needed.

Other security design differences exist between Windows and Linux, and only a few are given here as examples. A default Windows installation tends to omit key components that enhance security and others that destroy security are installed and activated by default. Many Linux distributions can be installed with different default security models and locking a Linux system is typically much easier. Another example is the included firewall capability; Linux has included packet filtering since the 2.0 kernel version. This firewalling capability has led many sites to use Linux computers as stand-alone firewalls to secure important networks (often, with a Linux system providing security for a network of Windows client computers). Windows XP, released about five years later, did include a firewall; however, in the initial firewall included with Windows did not work very well, though the upgrade to Service Pack 2 reportedly improves the firewall. Finally, the modularity of the Linux system (see Section 7) improves overall system security.

6 Available Tools and Utilities Are Also Open Source

Many a Windows user has downloaded a free screen saver, time synchronization tool or other free software only to find that software contained 'spyware.' Spyware is a type of software that monitors Internet surfing habits or worse; some spyware acts as keyloggers, trapping passwords, credit card numbers and other private information. The reason this happens as often as it does relates

to the Windows Software Deployment model (Section 4). Since Windows does not include a compiler, software developers must assume they do not have one and therefore deploy software as executable binaries. The source code is not generally available, so the presence of spyware components in a program is only detectable after the code is executed; after the code is executed once, the damage may be done.

With Linux, on the other hand, many tools, utilities and applications are Open Source Software (OSS). Open source, often confused with "free," simply means that the source code is available (even if the program is deployed as a binary, the source code is generally available). This means that many competent programmers can examine the program for integrity and flaws *before* it is run on their computers. This in turn generally implies that malicious programs get caught sooner.

Many Windows advocates argue that the only reason Windows has more problems with spyware and similar malicious programs is due to 'market share.' The argument goes that there are more Windows computers so Windows is a bigger, and more attractive, target for the malicious code producers. However, one major flaw in this argument is the Open Source nature of Linux installations. If Linux becomes a larger 'target,' that simply means even more eyes see the code of Open Source applications used on Linux. It is the Closed Source nature of Windows applications that make this type of spyware easy to deploy.

7 Linux Has a Modular System Design

One of the single biggest negatives going against Windows based systems is the large degree of integration of the Operating System with applications. Many people, apparently including the Microsoft marketing department, view this as a positive; rather, such a high degree of integration has significant security and stability consequences. For example, the Windows kernel is heavily integrated with the graphical user interface; there is no Windows without windows.

Linux on the other hand is very modular by design. Different components of a Linux system originate from different developers; each has their own specific design goals and focus on those goals. Further, each component is configured separately, generally by the use of text based configuration files. This modular design means that the Linux kernel is independent of the graphical interface, for example. The net result is that crashes and security vulnerabilities in applications tend to remain localized, rather than affecting the system as a whole.

The modular configuration system of Linux is somewhat daunting at first, as there is no standardized format for the myriad of configuration files that exist. But, there are some key advantages to this system. For example, configurations are not in a cryptic database (the Windows Registry). Reading and writing configuration information can be done by scripts or applications using simple text parsing engines; no special API is required to interface with the system configuration data. Script based configuration adjustments allow for such powerful techniques in Linux as dynamic routing or firewalling done 'on the fly' which

are considerably more difficult in a Windows environment.

Indeed, the shell itself is one of the most powerful Linux 'modules' that has been all but ignored in recent versions of MS Operating Systems (Windows is so extremely tied to the graphical interface that CLI use is relegated to nearly substandard status). Scripting of Linux systems provides a level of automation and control that is difficult to achieve in Windows. I've played around with the Windows Management Instrumentation (WMI) that uses Visual Basic Script as the scripting language; between poorly documented API's and object specifications and the work required to interface with the registry or other system resources, WMI *is* a useful tool but weak compared to a shell such as bash in Linux.

With bash, one can in a single line determine system memory, an allowed port with the firewall, change an email address in a dozen html files or any of a host of other useful tasks. In short, the Linux shells have been developed to make Linux administrators *productive*!

Finally, another aspect of modular design is the use of links in the Linux file system. Windows allows the use of shortcuts, but they are not links in the true sense. A shortcut is not evaluated by the OS as the shortcut target file, it simply acts somewhat like an alias for the filename. A link on the other hand is a true abstraction of a real file; if you open a 'link,' you actually open the file to which it points. Links are extremely powerful in maintaining a properly running system, especially if one upgrades application and system components.

One of the most important uses of links is to generate a 'pointer' to a library. Suppose an application uses the shared library mylib-2.3.so. Rather than hard-code the application to read this particular file, it is coded to read mylib.so, which is really a link to mylib-2.3.so. Subsequent upgrades of mylib can be made without having to recompile the applications that use it; only the link target needs to be changed. This allows one to have many versions of the same shared library on a Linux system simultaneously, which cannot typically be done on a Windows system. Applications designed to use whatever the newest version is call the link (assuming it always links to the newest version), whereas applications that require a *specific* version can be compiled to call that version rather than the link. This technique allows all Linux software to have it's required environment and one application cannot break another; this is the essence of modular design.

On Windows, dynamic library file names typically do not contain the version. This means one cannot tell by simple inspection what version a given library is. An application that requires a specific library version, say mylib.dll version 2.3, cannot coexist on a system with an application that requires a different version (such as mylib.dll version 3.5 or newer), since there would be a name collision. Here we see another disadvantage of both the software deployment models and Open vs. Closed source models of Windows and Linux. Applications on Windows typically cannot be modified by the end user, so one cannot recompile using a different file name for the necessary dynamic library.

8 Driver Stability

I've often heard Windows users claim "it just works." Sorry, but this has not been my experience, and I have not been known to try to use outlandish hardware. For example, I have one HP 990C Ink jet printer, and the Windows driver supplied by HP crashed numerous times per year; attempting to stop a print job invariably crashed the driver, and the only recovery was to reinstall the driver. Many times, while browsing the Internet using Internet Explorer, the modem driver would crash; this never happened after Mozilla Firefox was installed and used as the default browser on that machine, so clearly it was a problem not only in the driver itself.

I have done some work in a computer repair shop servicing Windows computers. Failures experience by customers were typically of three categories: virus/malware issues, driver issues and hardware failures. By far, the most common was driver problems. Windows based computers, in my experience, are just not stable platforms, but the culture of computing with Windows is to accept these inconveniences.

In contrast, I have numerous Linux systems running a variety of hardware; I've yet to have to reinstall a printer (for the same printer) or any other driver. Some people claim that Linux does not recognize their hardware, which may be true; however, there are Hardware Compatibility Lists for Linux that give known, good hardware. Incidentally, there are similar lists for Windows. I've installed numerous distributions of Linux on numerous computers, and I've only seen one hardware recognition issue at install-time (an old Version of Red Hat did not recognize the chipset on a new main board; the same thing would happen with installing say Windows 95 on new, 2005 era hardware).

9 Linux Configuration Makes Sense

About Linux, it is often said that 'everything is a file.' This means kernel and system parameters, device interfaces, etc. are accessible as text files. For example, one can enable packet forwarding in the kernel by editing the 'file' /proc/sys/net/ipv4/ip_forward. Though this is a system kernel parameter, it is read and manipulated just as if it were a text file.

Personally, I find configuration of Linux much easier than that of Windows. In addtion to the system parameters in /proc, Linux systems store application configuration information in one of two places. System wide configuration (that applies to all users) is stored in the /etc directory; per user configuration settings are stored in the user's home directory. A text editor is all that is needed to edit Linux configuration files (though graphical configuration tools do exist) and the settings in the various configuration files tend to make sense once one reads the file (or documentation). For example, to configure the Samba SMB server to act as a domain controller, one needs the line (among others)

Domain Master = yes

in the Samba configuration file. This is pretty clear, and the setting is right where one would expect: in the file for the SMB server called smb.conf.

This is in direct contrast to the Windows configuration system. Configuration settings in Windows are done either using graphical tools (such as through Control Panel) or by editing the Registry directly (which is very error prone). To illustrate just how confusing Windows configuration can be, consider this example. The author recently had the opportunity to set up numerous Windows XP computers for domain membership and as IPSec VPN clients. Several of the relevant Windows settings contain double (or triple) negatives with the options in "enable" or "disable" format. For example, one setting was

Disable machine account password changes, Enable or Disable

Setting this to 'Disable' means that machine account password changes are enabled. How's that for a mental gyration to perform at 3:00 am while configuring a system during 'off peak' time to be ready for the next normal business day? Another is the Disable Last Accessed setting for performance tuning the file system; one needs to set this to "0", meaning "Don't Disable." To me, at least, it is quite clear why Windows is difficult to configure properly for either proper operation or for security.

10 Linux Offers the Opportunity for Higher Productivity

I started my scientific consulting business in 2000. Since my business inherited my personal computer, the first DSB Scientific computer ran Windows 98. The purchase of a new computer came with Windows Millenium Edition (which did not last long, a month or two perhaps) and ultimately served with Windows 2000 Professional Edition. This computer operated in dual boot mode with Red Hat Linux, but did spend most time booted into Windows. In 2004, it was the last of DSB Scientific's computers to become 'full-time' Linux; it is now running Mandrake Linux nearly full time, and is the only computer in the stable with any version of Windows even installed.

The purpose of this brief historical account is to provide contrast in how productive this particular computer was when running as a Windows box versus running as Linux box. The role of this computer has not changed: it is a development platform and an office computer (word processing, accounting, email client, etc).

While running Windows, at least two low level crashes occurred per year that resulted in OS reinstalls. The computer ran Adaware to catch spyware, Norton Antivirus (annual subscriptions required to keep up to date), MailWasher antispam tool and required several hours per week just to maintain the OS in good working order. Printer and modem drivers crashed anywhere from once

per month to every couple of months. If I wanted either a hardware or software firewall, I would have had to buy one. Backups were accomplished using a simple copy of files to CD, but due to certain files being 'flagged' by the OS, certain directories could not be copied *en masse*. Therefore, backups were difficult to automate (note: the size of the backup space was too large for the Backup Tool that came with Windows; it always crashed). The bottom line was the computer itself was a productivity sink, not a productivity producer.

In contrast, the same computer running Linux has, on the other hand, become a tool that allows the author to get work done. No longer is simple, day-to-day maintenance of the computer a tremendous time sink. For example, the printer driver has not crashed once in over a year of hard use (note: this is not some obscure printer; it is a popular business class HP Inkjet). No longer are half a dozen 'utilities' needed to run full time (consuming CPU and memory resources) to protect my system from malware. The kernel level IP packet filtering makes a powerful firewall and was included (and installed) with the OS. Installation of the OS and secondary software was accomplished in about 1/3 the time with much less user interaction. Boot time is quicker than with Windows, and networking with the other systems on the network has been much more reliable. Backups are done with the tar utility that has 'complained' about neither the size of the archive nor the backup of 'protected' files in the user's home directory.

Conclusion

With this article, I hope I've shared some ideas that provide 'food for thought.' I know many people that run Windows and won't even consider Linux for a second (if they've even heard of it). That's fine: their machines, their choice. But for others, those wishing to shop for the best solution for their situation, I would hope the experiences of someone who has used both extensively will prove valuable.