A Live Report from Linux-land:
What a New User Should Expect

Paul E. Johnson
Dept. of Political Science
University of Kansas
Lawrence, KS 66045

pauljohn@ukans.edu
(913) 864-9086

Draft 2, Sept. 8, 1997

Not long ago, I became interested in simulation models that take into account the interaction of large numbers of individuals. I did some research and discovered that, at the Santa Fe Institute, Chris Langton and colleagues were developing a program called SWARM, which held out the promise of a general simulation language for multi-agent systems. To my lament, SWARM was built for UNIX systems, and I did not have a Sun Workstation or DEC Alpha Workstation. SWARM had been successfully installed on the Linux operating system, however, and I was intrigued to know more about it.

This essay describes some of the observations I made during my personal journey from the DOS/MS-Windows world to the world of Linux. I installed the Linux operating system on my home computer and ran some small simulations. The hard disk still holds a copy of Windows 95, and I expect it will stay there because I've not been able to wean myself from a few of the more useful Windows programs. But, after restarting the computer under Linux, I can run programs designed for the UNIX operating system, a powerful multi-tasking, multi-user system. And the real beauty of it is this: the UNIX operating system is very expensive, but Linux is **free**. That's right! I downloaded it from the internet for free (without violating any copyrights!).

It is not easy to install Linux. Even though recent package "distributions" of the operating system and software have made it easier, beginners should expect a fairly high number of frustrating hang-ups. During the "break-in" period, the user will often wonder whether the user or the system is being broken-in. I have often suspected that the truth is some of each. It is not effortless to go between Windows and Linux—it is necessary to restart the computer to do so. While going back and forth, one might become confused by the simplest conventions. Linux uses forward slashes (/) in directory paths, but MS-DOS/Windows uses back slashes (\).

The hardship is justified if one needs to run programs designed for the UNIX environment. The Linux operating system has other sorts of appeal. The user is not limited by the kinds of artificial restrictions that have so regularly plagued MS-DOS programmers and users. There is no 640KB memory ceiling to be negotiated, for example. Linux, like the UNIX systems that it emulates, is a multitasking system designed to run many programs at once (true pre-emptive multitasking, much better than Windows '95). Beyond the technical convenience, however, another important benefit of Linux is purely emotional: users gain a feeling of freedom, autonomy, and self-determination. A Linux system offers users the power to adjust and tinker with the "look and feel" of their computer.

My primary purpose in this essay is to describe what Linux is, why it is useful, and to provide a sketch of the installation and usage of the system. The presentation is aimed at an experienced MS-DOS/Windows user, not programmers. I don't write with the background of a computer programmer. Rather, my knowledge of computing is almost totally learned by the seat-of-the-pants in the practice of social science.[1] I hope to help others make informed decisions and save time and effort when they consider installing Linux.

What is Linux?

First things first. How do you say that? I've heard many pronunciations of "Linux." The person who created the Linux kernel, Linus Torvalds, pronounces it with his native Swedish accent in a way that sounds like "Lin'nux" (rhymes with "cynics"). If you want to hear him say it, browse my Linux page: http://lark.cc.ukans.edu/~pauljohn/linux/pronounce-linux.au. Some people say "Lī'nux" to make it rhyme with the long-i in the American pronunciation of the name "Linus". Whichever method of pronunciation you choose, people will know what you mean!

Linus Torvalds was 23 year-old college student at the University of Helsinki when he began investigating the possibility of creating a “UNIX-clone” operating system for the personal computer. The Linux operating system creates an environment in which one can run UNIX programs on the same personal computer that was probably purchased with the intention of using a Microsoft operating system. Linux provides the “kernel” of the operating system, which means it manages the “low level” operations that go on inside a computer: it gives programs time on the central processing unit, and it communicates with the hard disk, memory, and video display. In 1991 he announced his design and invited programmers around the world to join in the effort to expand the system’s capabilities. Linux has all of the appealing features of the Unix operating system. It transparently manages the activities of many programs and users. Unlike DOS, programs running under UNIX are not plagued by any artificial limits on the amount of RAM that can be accessed. UNIX has native support for TCP/IP (internet protocol) as well.

Linux responds in the same way as UNIX to commands, but it is legally and financially a separate thing from UNIX and companies that sell UNIX. One might refer to properties of the “UNIX/Linux” operating system, but the legal distinction should always be kept in mind. There are many different copyrighted UNIX operating systems (and kernels) sold by competing companies. There is a continual search for an industry-wide agreed-upon standard, but still there are minor differences among versions of UNIX (people who are in the know call them “flavors” and use the plural “Unices”). Readers might have heard of companies like Sun Microsystems (which sells Solaris, a UNIX OS) and Digital Equipment (which sells Digital UNIX, previously known as OSF1). Linux, on the other hand, is provided for free to users under the terms of the GNU General Public License, which is administered by the Free Software Foundation. The GPL is a long document, but basically it endorses the concept of freely-available software. If someone uses and improves a GPL-ed program, the GPL requires that the resulting source code be available without charge. To emphasize the difference, the FSF homepage uses the term "copylefted" to describe the legal status of such software. "Copyleft says that anyone who redistributes the software, with or without changes, must pass along the freedom to further copy and change it." (http://www.fsf.org/copyleft/copyleft.html). In addition to the Linux kernel, other vital programs, such as the GNU's editor "emacs" or the GNU compiler for the C/C++/Objective-C language family, are available under the GPL.

What is Linux like, though? How does it work? It may not be possible to describe an elephant to someone that has never seen a four-legged animal. If someone has never seen a computer, perhaps they will not be able to imagine what Linux is. But, if someone has seen a dog, one can describe the elephant by contrast (e.g., it is much larger, not so fuzzy, with a really long, tubular snout). I would pursue a similar strategy in describing Linux to someone who has used Microsoft DOS (disk operating system) and Microsoft Windows. For reasons that historians might debate, the Linux/UNIX operating systems have many similarities with MS-DOS and MS-Windows. (Since the UNIX operating system existed first, the readers might judge for themselves which one is a pale imitation of the other.) MS-DOS is like the UNIX console mode, or terminal screen. The user sees a "shell prompt." (A shell prompt is called a "command prompt" in MS-DOS.) Commands can be typed in, results may be written on the screen or into files. Instead of typing "dir" to see a list of files, in Linux one types "ls." Programs run in the console may use color and graphics, in the same way that DOS programs do—that is, with some difficulty. The Linux/UNIX systems also offer a graphical-user-interface (GUI), which is called the X Windows System, or X for short. Anyone who has found that a program runs only in MS-DOS, and not MS-Windows, or only in Windows, and not MS-DOS, will feel completely at home in the UNIX/Linux world. The same basic rule holds: Programs designed for the GUI (X) run only under the GUI, while *some* programs written for the nonGUI (the console) run under the GUI as well.

Aside from its ability to easily manage large amounts of memory and powerful CPUs, the Linux/UNIX operating system has important features that distinguish it from MS-DOS. Perhaps most importantly, a Linux/UNIX system is a multi-user system. When the system is installed, the first user created is the "root" user, also known as the "superuser." This is the system administrator, the user who has the responsibility for maintaining software on the system. The superuser is the only one with permission to install most programs or run programs that affect the way the system operates. The superuser can create other users and designate their rights. Each user gets a "home" directory, e.g., a person named fred might have a home directory named /home/fred. Each user is free to customize the way the Linux system responds to commands (more on that later). Each file has an owner, the only person that is allowed to control access to that file by other users. The owner can assign "permissions" which say whether the file can be read, written-over, or executed by the owner, a member of the owner's group, or any user on the system. One of the most vital pieces of advice to a new Linux user is this: Never run anything as root unless you absolutely have to. One should log onto a Linux system as an ordinary user, and if necessary, log on temporarily as the root/superuser by using the command "su root." This prevents traumatic accidents because ordinary users are not allowed to delete vital system files.

Only the most courageous do-it-yourself user will try to build a Linux system from the ground-up, even though the source code is available. Most people will use a "distribution" that includes the Linux kernel along with lots of other programs that are standard parts of Linux/UNIX. As far as I know, every UNIX operating system has a C compiler, for example.

Linux systems typically include the GNU C compiler. Any user can type in a computer program, use that freeware compiler to turn it into an executable, and then run the program. Distributions are getting bigger, including more utilities and handy programs. Whereas installation was once fraught with difficulties that might drag out over days or weeks, now one can use a CD from a distribution to install Linux in a day or so. In the present state of refinement, the major difficulties will be encountered in the preparation of the PC for installation of Linux (more on that later).

Once Linux is installed on a computer, it has a well developed “directory tree,” which means a hierarchy of directories. A program is run by typing its name, assuming that file is in a directory which is on the “path” that the system searches. Some directories will look familiar to people who used MS-DOS in the old days. There is a directory called /bin (short for “binary,” meaning “executable file”) that holds the most generic, universally necessary programs. (On my first Zenith computer, dos commands were stashed in a directory called c:\bin.) In /bin, one will find the programs that do basic things, such as copy files. If a program is installed on a Linux system, and it is freely available to all users on that system, it is often installed in the /usr/bin directory, or perhaps /usr/local/bin, or perhaps /usr/X11R6/bin. The hierarchy under /usr includes a directory that has the source code for various programs, including the Linux kernel, as well as libraries of shared resource files and document files. File systems differ from computer to computer because there are personal judgments that affect which directories are used for certain programs. Because of the potentially chaotic state of affairs, the Linux community began working on the File System Standard, a document which advises users on which directories ought to be used for which purposes.

Some of the Linux directories will look very unfamiliar to the DOS user, however. For example, my Linux system has a directory called /dosc. That directory is the C drive where Windows 95 is installed! If I list the contents of that directory by typing “ls /dosc” I notice that the subdirectories are the directories that I have created on the Windows side of the PC. Linux treats every device as a directory! Linux can read and write on a variety of file formats, including the file systems that came with MS-DOS through version 6 and Windows 95. As of this writing, the Linux community is still experimenting with the newest Microsoft file format, VFAT32.

The treatment of devices is most apparent in the directory called /dev, which houses an enormous list of items, ranging from /dev/modem or /dev/mouse to the puzzling /dev/cua0 or /dev/tty5. The way Linux/UNIX uses these directories holds the key to the portability of the UNIX programs. All devices—hard disks, video cards, sound cards—are treated as files in the directory structure. If the system wants to make a sound, it might send a message to /dev/desp, which represents the sound card. The way UNIX/Linux uses directories makes programs portable across platforms in a way that the Macintosh OS or Windows are not. A program written for one UNIX system on a particular kind of hardware is likely to work on a UNIX system that has completely different hardware, as long as the same directories and files exist. (If they don’t, users are instructed to create “symbolic links” which redirect program output from one location to another.)

Why consider Linux?

1. Some programs are available only for UNIX.

The simplest, most obvious reason to consider Linux is software. If you don’t have a personal workstation, you need Linux to run UNIX programs. There are plenty of programs that are written for UNIX that have not been ported (that's the word they use to refer to redesign) for Windows. Many scientific programmers work in a UNIX environment, so their programs run in UNIX. Windows versions might be nonexistent or delayed. I want to run the program called SWARM, and although plans exist to create a Windows NT version, in the here-and-now, Linux is the feasible option.

2. Linux can be less expensive.

Suppose you can choose between upgrading a PC to run Windows NT or Linux. Linux is definitely less expensive, both in terms of hardware (it requires less memory) and software. If one cares to, the Linux operating system can be obtained for free. It is not shareware. It is free. One can download a distribution of Linux from the World Wide Web. Documentation is free, too, on the Web. The biggest site for Linux related material is Sunsite (http://sunsite.unc.edu). In the directory called /pub/docs/HOWTO, one will find "HOW-TO" files that address many different aspects of the Linux system, ranging from installation of the system and software to details about specialized hardware. In the directory /mdw, one will find the homepage of the Linux documentation project. This will offer hypertext versions of many HOWTO files, as well as handy tip sheets and links to various Linux sites.

In bookstores or computer stores, one can find manuals with titles such as *Using Linux* (Tackett and Guner, 1996), which will provide CD-ROMs with a distribution of the Linux operating system. Such a book will probably be handy, but CD-ROM is likely to be out of date. The Linux kernel changes rapidly, as both major upgrades and minor fixes are created. For example, in 1997, the kernel was upgraded and "modularized." While Linux is running, the kernel can automatically start-up a module that allows the use of a particular printer or tape-backup drive. Modules allow the kernel to stay small—using minimal system resources—and still support diverse equipment. Very valuable updates, such as support for modules, imply that one should obtain a Linux distribution directly from a company that advertises on the internet.

There are a number of companies that sell Linux distributions. In the early years of Linux, the most complete package including the Linux OS and various programs was the Slackware package. More recently, companies such as Red Hat and Debian have come into existence to sell distributions of the Linux operating system.

I have experience with the Slackware and Red Hat distributions. I tried Slackware because it was on a CD that came with a manual. Within days, I discovered that it was outdated. At that point, I spoke with a computer consultant who said "forget Slackware. Get Red Hat. That's what Linus Torvalds uses." That turned out to be good advice. The Red Hat company makes available on the Web a full distribution of Linux, including many freeware programs. I downloaded about 150 megabytes of compressed software (but did not install it all). They make available a manual that one can download and print out. If I were to start over, I would have spent the $50 to order the newest Red Hat Linux package direct from the company. That would make the installation easier, provide a printed manual, and it would give me the right to bother them with silly questions. (So, if you are keeping track, I've spent about $100 so far.)

The main advantage of Red Hat Linux is the "RPM" system. The Red Hat Package Manager keeps each program or utility in a file with the RPM extension. The user can then use a program called RPM to install and remove packages. It keeps track of dependencies among programs, and will not let the user delete a program that is vital for other programs and it also will not install programs unless all of the software upon which they depend is also installed. The package system offers upgradeability, the easy way to add new versions and yank out old ones. The Red Hat company maintains a WWW site where users have contributed RPM packages for a wide variety of software. The Slackware distribution boasts its own package manager, but in my experience it was not nearly as powerful. I've not done in-depth research on the Debian distribution to make a statement about its relative power.

The Red Hat distribution will come with its own installation instructions. Those instructions are tailored for Red Hat and supercede any general HOW-TO information obtained elsewhere on the web.

3. Linux is nice to use.

There are handy little features that make Linux fun to use. For example, Linux supports long file names. Always has. But typing long file names can be tedious. A Linux user only needs to type the first few letters of a filename, however. By touching the tab key, the user causes the operating system to automatically search the current directory and complete the filename. Consider another handy feature. Linux remembers commands. If one needs to repeat a command that was issued a while ago, just press the up-arrow key repeatedly. The system will cycle through previous commands, showing them one after the other. Unlike MS-DOS, in which such commands are difficult to revise, Linux allows the arrow keys to move the cursor without erasing characters, so revising commands is extremely convenient.

These convenient features are available from the start. Others can be accessed when desired. For example, I like color-coded file listings. Each element in a listing of a directory can be color-coded to make types of files easy to spot. Text files are white, but executable files are bright green. Sound files are purple. Subdirectories are blue. The optional color listing is obtained by adding a flag to the ls command, so it becomes "ls –color". If the user wants to make such an option permanent, it can be done by editing a "hidden" file called .bashrc that is (probably) in the user's home directory. Files that start with a period do not show up in a listing unless the user types “ls –a”. (That’s the only sense in which they are hidden.) The .bashrc file is just one of many configuration files that are just waiting for the user to add a personal touch. These files are typically scripts that are accessed when programs run. No inscrutable interface or menuing system is needed. A simple text editor is all one needs to have virtually total control over the colors that appear on the screen or the locations that the computer looks for programs.

Like many new users, the customizability of Linux is as much a source of confusion as promise for me. One source of frustration was the term “shell,” which is absolutely integral to the understanding of the user interface. I first heard the term when I had trouble with my account on the DEC UNIX system that I use to read e-mail and run SAS jobs. The consultant asked “What shell are you running?” I must confess I was baffled. It never occurred to me that everything I did was occurring within a program that received my commands, conveyed them to the kernel of the OS, and then conveyed results back to me. That is a “shell.” MS-DOS, with its command line and such, is a shell. The big difference in Linux/UNIX is that there are many shells, some old, some new, and each of them uses its own terminology and each can be specially tailored to the user’s preferences. Almost all new Linux systems will use the “bash” shell, short for “Bourne Again Shell.” This is a modernized version of Stephen Bourne’s original UNIX shell. There are others, of course, such as the C-shell or the Korn shell. In MS-DOS, one can write a batch file—a list of commands—and make it executable by attaching the suffix .bat. In Linux/Unix, such files are called “shell scripts” and they are quite easy to prepare (see, for example, Matthew & Stones, 1996).

4. Each individual has a personal duty to resist the evil empire.

I was mad at Microsoft even before I knew of Linux. Problems with configuration of MS-DOS games that needed more than 640KB of RAM (like Wing Commander II or Front Page

Sports-NFL Football) drove me crazy. I got more upset when I noticed the indiscriminant consumption of hard disk space by MS products. Witness the bloating of Microsoft Word. When I started with Word in 1985, it ran off a single 5 ¼ inch floppy disk. Word for Windows '95 takes more than 15 megabytes. No wonder cynics call it "bloatware." In an effort to throw-in a function for every conceivable purpose, they created a monster. The same is true of other MS products as well. Recently, I wanted to run some programs written in C++. I ended up using Linux and the GNU compiler, which is clean and effortless. I went that route after I saw that Microsoft Visual C++ was so cluttered with bells and whistles that it demanded 50 megabytes of hard disk space.

The MS approach really perturbed me when I started studying the revised version of Windows 95 (the so called OSR2, or Win 950B). If you install the first version of Win '95, you are allowed the option of saying NO when asked if you want to install software for the Microsoft Network. Even if you say no, the installation drops a directory called \Program Files\The Microsoft Network on your hard disk. OSR2 does the same, but it also installs several megabytes of software for Compuserve, Prodigy, and AOL without permission. There's no way within the add/remove programs regime to get rid of unwanted files like that. I suspect that the average user has no idea the hard disk is crowded with them. (Lately I've noticed companies like Acer and Compaq are following suit, selling PCs with 50 or 100 megabytes of promotional videos hidden on the hard disk.)

But with OSR2, I found the shocking development that the installer slaps Microsoft's Web browser "Internet Explorer" (IE) onto the hard disk without asking. That bothered me because I prefer Netscape's Navigator and don't want to carry the deadweight of IE. Unlike other Windows features, such as Wordpad or the calculator, this program does not show up in the add/remove panel that would allow a root-and-branch removal. One can simply delete the IE directory, but not without some concern over whether a necessary program might be disabled. If one buys Norton Utilities II for Win '95, one will be dismayed to find that it also checks the system to find out if you've gotten rid of Internet Explorer, *and reinstalls it*. Why would Microsoft be so insistent on jamming IE down my throat? It is obvious that they want to discourage me from using a web browser from the leading competitor.

I understand the argument that pluralism is not necessarily a good thing in the computer world. The diversity of operating systems may create inconvenience. However, these costs are far outweighed by the advantages of competition--rapid program development and low cost. In Linux, I have much more freedom to pick-and-choose programs that I want to run, and almost all of them are free. I've never seen a Linux program that tried to edge out a competitor by taking up disk space that was intended for it.

Is Installation a Hassle?

The installation process was a terrible hassle for me. The experts might say, given recent improvements, it should take an hour or two to install, but I'd bet that those estimates are made for users who know what they are doing. New users will take wrong turns, get confused, read the wrong document, and waste time.

The first step is to ignore the boy-scouts in the Usenet groups who advise the new user to try to compile Linux and other software for themselves from scratch. (Look in the hierarchy comp.os.Linux.* for such opinions). Instead, buy a Linux distribution that includes the OS, lots of software, and good instructions. After comparing distributions, I chose Red Hat. Second, buy your distribution directly from a company that advertises on the internet. This will guarantee you the newest version.

The installation process for Linux has been simplified somewhat through the past two years. Still, some fundamental details must be attended to. First, one must find hard disk space for the Linux system. The minimum space needed is about 100 megabytes, but I would recommend a gigabyte or so. If Windows is currently using up all of the hard disk space (that's typical), one must either install another hard disk (get the same brand as possible to simplify installation) or shrink the amount of disk space devoted to Windows. Microsoft does not make this easy. Straight-laced consultants advise a person to wipe out the whole hard disk and then create a new partition where Windows can be reinstalled. One creates a diskette from which to boot the MS-DOS operating system, and from that disk the FDISK program is run to destroy the old Windows partition (and everything on it) and create a new, smaller partition. Since reinstalling Windows is a major hassle, one can try programs that can nondestructively shrink a partition. I had excellent experience with a program called Partition Magic 2, which ran in DOS--off a floppy disk--in a simple and transparent way. There is a freeware alternative called FIPS, but I have not tried it. I cannot testify from experience that these programs work on the newest MS file system, VFAT32, but they worked fine with VFAT (that's what the first version of Win '95 uses).

Once the user has Windows in its own partition, there should be space to create two partitions for Linux. In order to create those partitions, it is necessary to start the Linux operating system, usually with a specially-created Linux boot disk. Distributions of Linux will offer different methods installation. In Slackware, for example, one is instructed to create a boot disk for Linux, and then one uses a program called "rawrite.exe" to copy a pre-made, generic Linux kernel onto a second disk. When the boot disk is inserted, and the computer restarts, the boot disk will be activated and then the system will ask for the second disk, which it calls the "root disk." When the central processing unit finds the kernel on the root disk, the Linux system will be activated. The system will ask for a password that is to be used for the superuser, and then it may automatically start the Linux version of fdisk.

Beginners should create two partitions. One partition can hold the Linux software itself, the other is a "swap partition." As in Windows, the swap space is used to store information when memory (RAM) is full. The swap space in Windows is typically held in a file called Win386.swp, and a user can put that file in its own partition to protect other files from it. The Linux installation recommends that the swap be put in its own partition. The swap partition should be about twice the size of the physical RAM.

Some installation manuals advise new users to create multiple partitions and divide the Linux software among them. This is a valuable practice for experienced users, especially for administrators who are creating very large systems to be used by hundreds of users. For beginners, it will create more trouble than it is worth.

The installation process then moves on to select software to be installed. If hard disk space is plentiful, a person ought to install everything. With a distribution such as Red Hat, it is easy to delete packages that are not needed in the future when space gets tight. On the other hand, if one is selective, Linux can be installed in a very small amount of space—certainly less than 100 megabytes—and packages that are accidentally omitted can be added later. The reason to install everything is learned through pain and suffering. Like MS-DOS, the error messages that one gets from Linux/UNIX software are not very clear. If you find a piece of software on the internet, and nobody in the Red Hat user community has made an RPM for it yet, you may try to install the software yourself. That means you download it, decompress it, and then and install it. The installation process typically requires one to edit one or more configuration files to specify directories where the program can find pre-existing software. Then the “make” program is used to convert (or “compile”) the source code that was downloaded into an executable program that is copied into the file system. When the program is used, it may crash and give only vague messages about what went wrong. Several times, this has happened because I did not have some requisite software installed that the program needed. Of course, if one only uses software that comes in the RPM format, this problem does not occur because the RPM program will not install a program unless all software upon which it depends is already installed.

The final critical decision concerns the method that will be used to start Linux. Most distributions assume the user will install LILO, a program that is written onto the Master Boot Record of the hard disk. When the computer is booted, the user will be asked whether Linux or Windows is to be started. I was cautious about this because I had heard that some people damaged their Windows partition by installing LILO. There are two alternatives. One is to use the boot disk to enter Linux by restarting the computer. That is inconvenient. The alternative is a program called “loadlin” that is included—but curiously not documented—in most distributions. Loadlin is an MS-DOS program that sits in a directory called c:\loadlin on my PC. In that directory there is a copy of the Linux kernel, which by convention is called zImage (on some systems it is vmlinuz). To start Linux, I restart in MS-DOS mode, then type these commands:

```
cd c:\loadlin
smartdrv /c
loadlin zImage root=/dev/hda4 ro
```

These commands can be put in a batch file for convenience. This restarts the computer in Linux. I’ve learned that there is another major advantage of loadlin that people with computers with

Plug-and-Play technology ought to take into account. PNP causes trouble for Linux systems because the DMA and IRQ assignments might be difficult to ascertain. If you use loadlin, the computer keeps track of those assignments and Linux can use them. Until I did this, I was not able to use my PNP sound card while in Linux.

What can you do "out of the box"?

When Linux is first started, the user sees a prompt to logon, and when that is finished, the screen will show the "shell" prompt for the bash shell. The commands mv, rm, cp, are Linux/UNIX commands to move, remove, and copy files. The commands mkdir and rmdir will create and remove directories.

Every Linux distribution that I know of will have the staple UNIX programs, such as the mail program "pine," the editors emacs and vi, the C compiler, some game programs, and other utilities. Even if a PC is not in a network, it still has a mail facility. Why? So the system can send mail to the root user, of course. For example, the Red Hat distribution installs a program called crontab, which runs system maintenance jobs at predesignated times, perhaps weekly or monthly. Cron updates indices of manual pages, for example. If something goes wrong when the crontab program tries to run, it sends an email to the address root@localhost. If one logs on as root, and starts the email program pine, those error messages can be viewed.

New users will get their feet wet in the console. After the user is able to copy files, move them, rename them, edit them, and the like, the next major challenge is X windows setup. Linux distributions typically include the free version of the X Windows server known as XFree86. It is called a "server" because it controls access to shared resources, such as the display screen, keyboard, and mouse. The programs that are sharing the resources, the "clients," are the programs we use to take advantage of the graphical user environment. The current (brand new!) version is 3.3. The main chore in installation is to set hardware specifications in a file called XF86config. The XF86config is a big file where the user can read about the various specifications for fonts, mouse type, video card, monitor refresh rates, resolution, and the size of the virtual desktop. That file will typically be found in the directory /etc or /usr/X11R6/X11. The boy scout in me says that every user ought to open up that file and edit it like a soldier. But the consultant in me would tell the user to try to use the configuration program "Xconfigurator." That program will take the user, step-by-step, through a selection of settings. That will write the XF86config. The user starts the X Windows program by typing "startx", which is in fact a list of commands that tell Linux to start the GUI and access various start-up specifications, including XF86Config. If X does not start, it means some settings in the XF86Config file are wrong. Users should be cautious about editing settings because there is the possibility of damaging the monitor by improperly setting refresh rates.

Setting up Xfree86 is a nightmare for many people who have hardware that is not yet supported. If the video card in the machine or the monitor is not recognized by Xfree86, the alternative is to get a different video card or to buy a commercial X Windows server, such as Accelerated-X or Metro-X. Either of these will cost about $100. In my experience, these commercial X Windows servers are considerably more simple to setup.

Once X starts up, what will the user see? One or two "xterm" windows will be opened automatically. These will look like consoles, except that they are inside windows that can be stretched or dragged like windows in other programs. They have little buttons on top that can close them or shrink them to icons. (Come to think of it, they look like MS-DOS boxes in Windows '95). Commands can be typed inside these xterm windows. Beyond that, it's hard to say what the appearance of X might be. The appearance depends on something called a "window manager." The window manager will set the color scheme, the style of the windows, and it will also create method for launching programs that it knows about. Red Hat version 4.1 includes a number of window managers and uses the one called "fvwm95" by default. This window manager looks a lot like Windows '95, featuring a start menu button and taskbar. Someone who is a Windows '95 user may feel at home, but there are other window managers that can be tried. My favorite is the Afterstep window manager.

There are some programs that are not distributed with the Red Hat package, but they are available in RPM format on the Red Hat site. In my opinion, any new X user's first step ought to be to get a copy of the newest version of a program called TKdesk. TKdesk is the like the Windows Explorer and it will save the user much trouble. For example, files obtained on the WWW are often "tarred" and "GNU-zipped." If a person can remember the UNIX command to untar and unzip the file, no problem. If you have TKdesk, however, you don't have to remember all that syntax. Built into TKdesk are scripts that execute common commands such as this.

Problems that Drive New Users Crazy

I have an interest in operating systems, but most users do not. They would rather just write their papers and read their mail without knowing anything about how the OS works. Unless one has an administrator who is in charge of installing software and maintenance, some knowledge of the OS is necessary, whether one is an MS-DOS or Linux user. It is definitely more time consuming to manage a Linux system. My experience is that many of the problems that burn up time are trivial matters that can be remedied easily with a little warning. In that line, I've prepared a list of problems that might crop up.

1. Unsupported hardware.

Even a savvy computer user can make the mistake of assuming that Linux will work with their hardware. Since Linux is freeware, there is no company standing behind it that you can yell at when your video card will not work with X windows. There are many dedicated, hard working Linux programmers who try to write drivers to suit hardware, but sometimes manufacturers do not cooperate very well with Linux types who need technical specifications in order to devise software. Some users will be forced to buy different hardware, get specialized commercial Linux software, or wait for freeware to become available. I waited 6 months before Xfree86 supported the Matrox Mystique video card. New users should always check the "supported hardware" list provided by the manufacturer of the distribution they plan to use.

2. ./

Linux manuals warn the new user about this, but still most fall into the trap. I thought my brand new Red Hat Linux had something fundamentally wrong with it. I could run programs to copy files, erase files, and the like. If I typed "pine," it started. However, sometimes I could type "ls", see an executable file, but when I tried to execute that file by typing that name, the error message was "command not found." I didn't have the same problem in Slackware. The explanation for this peculiarity goes back to the idea of a "path," with which most MS-DOS users are familiar. A path is a list of directories where the CPU looks for a program when a user types in a command. If a program is not in the path, the error message "command not found" is issued. If you happen to be in a directory that is not in the path, then you need to type ./ before your command, or else the system will not find the file you want. Many experts advise against putting "." in the path for security reasons, and recommend that you get used to typing ./ before commands. I see this new user problem crop up at least twice a week in the Usenet discussion groups, where new users post hysterical messages and get a mixture of responses, usually "RTFM" (which stands for Read the 'Fine' Manual) or "try ./".

3. These man pages are terrible.

When someone writes a program for Linux/UNIX, they are supposed to prepare a manual page ("man page," for short) to help the user. These pages are accessed with a program called man which is included in the Linux distribution. Man pages are clear, readable treatments that work from the ground-up, explaining for even the most primitive of user what exactly needs to be done. NOT! As one text observed, "Many manual pages are cryptic, misleading, poorly organized, or erroneous in subtle ways" (Abrahams and Larson, 1996, p. 24). Some man pages are clear and helpful, however, and if a command won't work, it is completely understandable when a consultant asks, "did you check the man page?" Most man pages lack clear examples that could easily guide the user through a difficult learning process.

4. Is the manual wrong?

I have often wondered if the manual—the man page or another document—is wrong. Manuals are usually correct, but not always, and when things don't work there is the lingering doubt. Should I ask? There is a clear emphasis in the Linux/UNIX community on reading and studying manuals (remember: RTFM). There's no expectation that anyone can sit down and make a computer work without studying. It is rude to ask a question that is answered in a HOW-TO, readme, man page, or other public document.

When the man page does not help, the next logical step is to look for a "readme" file. Many programmers will include readme text files with their programs. Readme files, like man pages, are not inspected by professional editors who assure clarity and completeness. I was astonished when I installed a programming library whose readme file had a six step sequence, but the commands in step number 5 were accidentally omitted. Those steps actually did the installation—copying the files into their directory and linking with other libraries.

There is an omission in the Red Hat 4.0 guide for revising the kernel that caused me some confusion. Linux users will usually do well enough with a kernel that comes “out of the box,” but the time will come when they want to custom tailor their kernel for their PC. The Red Hat Linux User’s Guide for version 4.0 spells out the steps very clearly for compiling a new kernel and configuring new modules. However, it neglects to mention that the old modules should be deleted from the directory lib/modules before installing the new ones. If one does not make that critical step, each startup of Linux brings a long string of errors about “unresolved module commands.” Even after making that fix, I still get a startup error message "modprobe can’t locate module net-pf-5”. One user has assured me that is not a serious problem and it relates to support for Appletalk networks (which I do not use).

Any Linux user has to have a rough-and-ready attitude about life’s little problems. In ninety-five percent of cases, the mistake lies in the user’s action, not the manual. Lest one think that this kind of problem is limited to Linux/Unix systems, I have had similar problems with MS-Windows and MS-Windows programs. For example, the server installation instructions that my department received for SPSS 7.5 gave the wrong directory information for the network installation, causing the installation to fail.

5. Do I have the right manual?

Most of the time, the manual is not wrong, but it contains information that the user is not able to decipher. For example, I wanted to run a program called “VH-man2html.” That program allows a Web browser such as Netscape to peruse man files and it enables hypertext links among man pages for easy browsing. Before Netscape can do that, however, a “web server daemon” must be installed. Think of the web server as something that runs for the root user and allows programs to access to files stored on the web site "localhost" that runs on the PC. The man page for VH-man2html is one of the few truly excellent, helpful man pages that I've found, and it recommends the user install a freeware web server called Apache. There is an RPM for Apache that installs it. In order to learn more about Apache, I studied the man page and the documents on the Apache web site. The manual page describes the configuration process. After that, the user is instructed to type "./httpd start" to bring up the server. That worked fine for me, but only the first time. I restarted the computer the next day, and when I tried to start Apache, I got an error message indicating that the port that the server wanted to use was already occupied. Why? After much frustration, I learned that the RPM installation process had added Apache to the list of programs that run automatically when Linux is started. I was trying to start Apache a second time, and it was conflicting with itself. After I discovered this, I browsed the Red Hat web site and found instructions that explained this quite clearly.

Here is another example of the same kind of problem. Experts will often recommend that new users should read available documents, such as *The LINUX User's Guide* (Greenfield, 1996). This is an excellent and generally helpful manual from the Linux Documentation Project that can be downloaded in postscript format and printed out. The guide has many very useful pieces of information, but one bit really drove me crazy. I wanted to change my X Windows manager. The *Guide* gives advice for the generic Linux system that existed before Red Hat: edit the file ".xinitrc" to tell the X Windows server which window manager to use. Such a file should be in the user's home directory. After noticing that I had no .xinitrc file in my home directory, I found out that the Red Hat approach calls for me to edit a file called .Xclients in my home directory that designates the window manager. Little details, such as mismatches in filenames, can bother the new user.

The experienced users usually react with disgust when I complain about problems like this. Why should I expect a man page or LDP document to know what the Red Hat installation process did to my setup? Man pages are written in general terms by the authors of UNIX programs. The program's author is not aware of any installation decisions made by the author of the RPM. After a few incidents like this, I've concluded that one ought to check for documentation in a logical sequence. First look at the Red Hat manual, then the Red Hat website for tip sheets or to peruse the archives of the Red Hat mailing lists. Then check the readme file that comes with the program, then the man page that comes with the program, and so forth. When I was trying to configure dial-up networking (point-to-point protocol, or ppp), I was very frustrated. The HOW-TO for PPP is available on Sunsite, of course, but it is a rather complicated document. Most of it is designed for network administrators who are creating terminal servers, not for people who want to dial up. I stumbled through the Web for half-a-day, downloading documents and programs that had hints about setting-up dial-up access. After I screamed about this, the experienced users shook their heads. On the Red Hat Web site there is a tip-sheet giving step-by-step instructions. All that one needs is an installation-specific set of commands. To save others from this, I created a manual of my own (http://lark.cc.ukans.edu/~pauljohn/linux/PPP-Guide.html).

6. & and the lingering dead X windows.

X Windows always opens up an xterm or two when it starts. A user can type a program's name at the command line in that window and the program will start. A console program will run inside that xterm, but a program designed for X Windows will typically open its own window to display itself. If the user shrinks the program window to an icon, the xterm is still sitting there. It will not accept any commands, because it is acting as the foundation for the program that is running. This causes trouble because the number of open xterms proliferates and the user can easily lose track of them. If the user shrinks a program and the xterm from which it started, an excess of icons quickly develops. If an xterm is closed, it causes the program that was started from it to close as well.

If a program is started with an & after it, such as "TKdesk &" then the TKdesk program will run it will release the xterm from which it was started, so more commands can be typed into the xterm. Some programs can still be accidentally killed when the xterm from which they started is killed, even if they start with the & option. A utility called "nohup" (short for no hang-up) can be used to address that problem.

7. Permission problems.

When I ran some computer simulations in Linux, they were written to a directory called /home/pauljohn/simulations. Windows 95 cannot access the Linux file space (one more reason to call it the evil empire!), so it is necessary to copy the files from the Linux space into the Windows file system. So I issued the command "cp /home/pauljohn/simulations/result* /dosc/temp", which should have copied all of the files starting with result onto the Windows part of the hard disk, in the directory that shows in Windows as c:\temp. I tried that repeatedly, there were no error messages, but it did not work. Then I discovered that a user such as "pauljohn" cannot write on /dosc. One can reconfigure the setup to change this, but the standard setup allows only the user named "root" (the superuser) to copy files onto the Windows partition.

The problem with permissions comes up in many strange places. Remember that each file has a permission specifier which says if the file can be read, written on, or executed, by the owner, a member of the owner’s group, or any user in the entire system. If a user runs a program that tries to access a file that cannot be written on, a program will generally crash. This can have important implications when it comes time to play games. (A very important time, I must add!). I installed a game called xgalaga that runs under X windows. This is a clone of the classic 1970s arcade game “Galaga.” It comes in an RPM called: xgalaga-1.6c-3.i386.rpm, and installation is as easy as signing on as root and typing “rpm –Ivh xgalaga-1.6c-3.i386.rpm”. Then, remembering the advice that one should not run as the root user more than necessary, I logged on as a user. When I typed xgal, I saw a pretty game screen, but then the game crashed with this message:

> $xgal.sndsrv: Couldn't open DSP /dev/desp

> $Broken pipe.

Why the problem? The file /dev/desp had permissions that allowed it to be written only by the root user. The command “chmod 777 /dev/desp” gave all users—in any group—the power to write on that file. After that change, any user on my system can run xgalaga.

8. Invisible carriage returns.

If one uses a Windows based web browser to view a text file, and then saves it, it will usually be corrupted by the addition of a carriage return at the end of each line. This will drive anyone crazy because most Linux/UNIX programs will not show the carriage returns, but the kernel objects to them. A script with hidden carriage returns will crash, with the message “command not found”. I burned up hours of time trying to debug a script until I, by chance, was using an editor called “nedit” to look at the file and the source of trouble, <cr>, appeared at the end of each line. Windows users should not save files after viewing them. Rather, use the “save link as” option that appears before a file is opened.

9. We are still working on that (or, “that’s a feature, not a bug!”).

The beauty of the Linux culture is that the source code is freely available, so anyone who wants to invest the time can try to fix a glitch. This is completely different from the MS-DOS world, where only the “official” MS employees can fix things. The flipside, however, is that there are fewer “gate-keepers” that stop mistakes or bugs from sliding through. But they are usually caught quickly, fixed, and made available for download.

I recently had an experience with Redhat 4.2 that reminded me of the frequent revisions of software. In the Redhat 4.2 package, the RPM system was upgraded to version 2.3.11. It worked fine, except when I asked the RPM system to verify all installed software, the rpm program “crashed” with a message “Segmentation Fault.” There is an RPM mailing list and web page within the Redhat site. I learned that RPM had been upgraded to version 2.4.3. If one browses the mailing list enough, one will find comments about how to install the new versions, but, to my surprise, instructions are not included with the new versions of rpm (which are available on ftp://ftp.rpm.org). After some guesswork, I installed 2.4.3, which solved the segmentation fault, but then it generated all kinds of new errors when installing packages. The authors announced that most errors could be ignored or worked-around, but the most fundamental problem was unaddressed. RPM packages created with version 1 of the RPM software could not be installed with version 2.4.3 of the RPM package! Well known commercial software, such as Red Hat Motif, could not be installed without reinstalling version 2.3.11. A few weeks later, after a few revisions, RPM version 2.4.6 arrived, which can install version 1 packages, does not crash, and produces only the occasional error message.

People who have only been end-users of MS products are probably horrified by experiences like that. But people who have been using Windows ’95 for a while have seen exactly the same evolution. The only difference is that bug fixes from MS seem to emerge much more slowly and with more baggage on them.

Final Words

There are some Windows programs without which I cannot live. I'm used to word processing and news reading in Windows and I suspect I would have a hard time going one-hundred percent into the Linux system. The rapid growth of commercial offerings for Linux will probably address these issues. A version of Word Perfect for Linux is available, as are office products like the Applixware Suite from Red Hat and Star Office, which is included in the Caldera office package. It seems to me that the main long-term blockages against a total conversion to Linux would be compatibility with old data bases and document exchange with other users. The frequent changes in format for MS-Word documents may add some fraction of functionality, but they seem designed to block interoperability of MS-Windows with other operating systems. Another problem is that the UNIX versions of popular software, such as *Mathematica*, is more costly for UNIX than Windows.

My interest in Linux was originally driven by a very practical need to run software designed for the Unix operating system. As I have learned about Linux, I've experienced the joy of its power (particularly multi-tasking) and customizability. There have been many hang-ups. In retrospect, I was naïve to think that the transition from one OS to the other would be easy. But, all-in-all, the transition was tolerable.

If a colleague wants to run a program written for the UNIX OS, there's no reason to discourage them from installing the Linux operating system. After the hard disk is partitions, Linux does no damage to the partitions that house other operating systems.

References

Greenfield, Larry. 1996. *The LINUX User's Guide*. Linux Documentation Project.

Matthew, Neil, and Richard Stones. 1966. Beginning Linux Programming. Birmingham, U.K.: Wrox Press.

Tackett, Jack Jr. and David Gunter. 1996. *Using Linux*, 2ed. Que.

Abrahams, Paul W. and Bruce R. Larson, *Unix for the Impatient*. 1996. New York:Addison-Wesley.

---

[1]At the risk of sounding like grandpa at a family picnic, the first computer I can remember using was a Honeywell that took punch card input. While in graduate school, I used IBM mainframe computers with operating systems known as Wylbur, Music, and CMS. The terminals that were used to access these systems were replaced by personal computers in the mid 1980s. The IBM mainframes (and their operating systems!) were phased out, and we were introduced to the UNIX operating system which ran on less-expensive, but more powerful computers that replaced the mainframes. I learned some UNIX at that time because, as DOS users, we were able only to telnet to the UNIX system and run UNIX programs in terminal mode. In 1993, I learned about the Winsock, which allows Windows software to access mail or files on a UNIX system without noticing that they are reaching across operating systems, so I forgot most of the UNIX commands that I had learned.