by Peter Collinson, Hillside Systems

JANE MARINSKY

Automation

Two years ago, I wrote an article that attempted to analyze what was lacking in the approach to computing promulgated by Mr. Gates' company. The article was called "The Trouble with Windows" (*SunExpert*, January 1997, Page 26), and it attracted hate mail from several people. I had mail from people saying that what Microsoft Corp. generates is truly wonderful, and I shouldn't be criticizing it. I had mail from people saying that the article was heavily biased in favor of Microsoft, and why on earth was I writing it in–that holy of holies–a magazine devoted to UNIX. The mail was split pretty evenly, so I took it as a sign that the unbiased path I had attempted to follow was successful to some extent.

I started to write the article because I perceived that UNIX folks are very good at simply dismissing Windows products with a throwaway line. How often have you heard: "Windows is just ****"? (It's my pleasure to allow you to substitute your own favorite expletive.) This type of summary dismissal seems all too common and means there is no widely spread, well-understood set of rational reasons for rejecting, or even questioning, what happens on a Windows machine. I guess that when pressed, techies do have a bunch of reasons, which mostly boil down to reliability and code integrity. But it always seems interesting that these issues don't seem to worry the people who make the purchasing decisions. Perhaps the two groups don't talk.

As time has passed, it has become more fashionable to attack Mr. Gates' company. However, the current attack wave is based on the trading methods that have been, and are being, employed. Surprise, surprise, suddenly Mr. Nice Guy is really Mr. Nasty, and what's more, the so-called "Halloween" documents reveal that it's all intentional nastiness. (Just in case the Halloween documents have passed you by, they are internal Microsoft memos annotated and placed on the Web by Eric S. Raymond, author of "The Cathedral and the Bazaar," you can find the URL at the end of this article). Actually, I became somewhat more aware of the tactics adopted by Microsoft (and others) by reading Jerry Kaplan's account of the rise and fall of the Go Corporation: *Startup: A Silicon Valley Adventure*, (published by Penguin Books, 1996, ISBN 0140257314), which I read in one rush on a plane.

I am sure nobody can deny that some of the reason for the success of Windows is the point-and-click nature of the beast. Take an example: Microsoft Word has ousted Corel Corp. WordPerfect, although WordPerfect was the market-leading word processing package at one time, and certainly had a head start. My guess is that people have moved to Word because it is more mouse-oriented, and the learning curve to get simple things done is shorter. I confess that I haven't looked at WordPerfect for some time, but will undoubtedly do so again because it's now available for Linux. Word began to take over because WordPerfect inherited the magic keystroke approach to editing that was (and is) common in UNIX text editors. The observation then, is that most users don't want to train their body to use a magic keystroke to achieve some action, they would rather point and click with a mouse. The problem is the keystroke is a "hidden action," and you need to remember it, whereas pointing with a mouse requires little or no training.

However, as I attempted to illustrate in my column two years ago, the mouse-based approach doesn't scale. People are forced to do complex tasks using combinations of mouse movements, and there's no simple way to automate these. My view is the Windows product line has generated a large class of people who are essentially controlled by the system, their gainful employment is simply to make the mouse move. Being a slave to the computer runs counter to my personal philosophy: The computer's role is to do the boring tasks. The message to managers and users should be that much time is wasted doing things that are better done by the computer, and by "better," I mean faster and more accurately.

Challenging Microsoft

Well, Linux is now being hailed as the world's "last best hope" to provide computer users with some viable alternative that places control of their computers back into their own hands. I hope so. I like the ability to understand and fix the bugs on my systems. However, I think the media take on Linux at the moment is "good as a server, bad on the desktop." The reason for this oft-heard discrimination against UNIX systems has always mystified me. For an illustration, look at what Linux can do straight out of the box. You can pay $50 to Red Hat Software Inc. for a copy of its Linux release and receive a system that requires no further software expenditure. It comes on three CDs with a plethora of editors (including WordPerfect), an excellent graphics program (GIMP), full Internet connectivity with a wide range of user applications, database support (both open source and commercial), a complete programming environment encompassing many languages…the list goes on. Alternately, you can pull it over the Web for nothing, but then you don't get the commercial offerings.

Still the media says: "Not for the desktop." I am sure Linux is branded with this tag because some of its applications date back to early X11 releases and need to be made considerably better looking. Also, the applications that people apparently want need more work. There's a columnist who writes for a U.K.-based magazine that has "converted" to Linux from Windows, but in his most recent column, he was complaining about the lack of a nice-looking file manager. Well, I must confess that I haven't looked for a file manager on Linux and I never use the one I have running under CDE on Solaris.

The Linux world needs to put lots of work into making the visual desktop easy to install, easy to use and, above all, better looking. There are signs that "being better looking" is being tackled. The Red Hat distribution unpacks with several X11 window managers that present visually pleasing and usable environments. There are more window managers available on the Internet. As I said, work is still needed in the applications area. Many open-source applications use the Athena widget set, which has a dreadful look and feel.

The aim of making things easy to install is also being tackled, and again there is some way to go. The UNIX world needs to learn this trick from Microsoft. One thing Windows 95 did was to sort out connecting to the Internet with a modem. It used to be dreadfully difficult to do this. More often than not, it turnkeys on Windows 95. You are probably sunk when it fails; but when it works, it works very easily.

The UNIX world needs to get enough awareness to convince hardware manufacturers that they should supply information to Linux developers in the same way they do for Microsoft. For example, my machine that runs Linux also runs Windows NT. The manufacturer of my graphics card supplies an NT driver for its product and this driver knows about my monitor. I don't have to worry about setting up obscure configuration files. To be sure, X under Linux works on my system, using a reasonably simple configuration program. But it is supporting a generic monitor, so it does not run the screen as well as NT does.

Linux is now being hailed as the world's 'last best hope' to provide computer users with some viable alternative that places control of their computers back into their own hands.

So far, as a newbie Linux user, I have encountered no great surprises, or perhaps I should say, only pleasant ones. Linux appears to support the mix of System V and BSD commands that I am used to on my Sun running Solaris. If you use Solaris, you will feel quite at home on Linux.

Getting the Job Done

I have wandered down the Linux path somewhat more than I originally intended, and perhaps you are sitting there wondering about the starting point for some of the stuff mentioned above. What does that idiot mean about automation? I just sit at my terminal using my vendor-distributed system to get my job done. How do I get to the nirvana where my job will be easier?

Step one is realizing that it is possible to create your own scripts and tools to automate your work. You don't need to be a programmer, all you need is a willingness to learn. At the outset, you'll need the ability to use a text editor. There are various simple text editors around. If you use CDE on your Sun, then point the mouse at the screen background, click your right mouse button and you will be presented with a menu that includes a simple text editor. You can find another text editor by opening a shell window and typing `textedit`.

Step two is understanding the restrictions. If you never use a shell-based environment to talk to the system and want to create your own visual applications, then the job is harder. It's not impossible, but harder. Most UNIX programs are designed to work in the shell environment. If you use CDE, you can group these commands into actions and make those actions work by clicking from the desktop. The best starting place to tailor your environment is to use the UNIX primitive commands to create other more personal scripts that are initiated from the shell.

Step three is recognizing that a proportion of what you do is repetitive and that spending time on automating these tasks will ease your workload. The most normal course of action is to type a bunch of commands, realize that you did very nearly the same actions the day before, but say "OK, I'll think about automating it tomorrow." Tomorrow never comes. The next day, the same thoughts flit through your mind. Another common trap is "it's complicated, and it's only a one-off task." Well, one-off jobs are always done more than once.

Recognizing that you are doing a repetitive task is not easy, especially when you are concentrating on something specific. Look for common command sequences and don't worry too much about the files to which they are being applied.

For example, I tend to keep my personal archive of software online. The archive contains code I have written and all the publicly available programs I use. Of course, there is never enough disk space to store information that I want to access for a short period. Also, I want to minimize the load on my system backups. A complete dump of my entire system currently fits onto one (large) DAT tape, which is convenient for a number of reasons.

I want to find some way of storing file trees in a compressed form on the disk, so I know where they are and can access them as needed. On UNIX, there are two tools that take a file system subtree and create a single file. The commands, `cpio` and `tar`, were designed to drive tape units for archival purposes. However, on UNIX, a tape device is a file, and the programs can also write the same information to a file on disk. I tend to use `tar` because that's what I am used to.

To store my files, I can take all the files in some portion of the file system tree and create a `tar` archive, and then compress that archive to save space. To create the archive, I'll type

```
$ tar cf dir.tar dir
```

where `dir` is the name of the subtree I am archiving, and `dir.tar` is the file that the data will be packed into. I can now compress that file. These days, I use GNU's `gzip`:

```
$ gzip dir.tar
```

The command takes a file, compresses it and renames it by adding the suffix `.gz`. Actually, these two commands can be combined into the following one-liner:

```
$ tar cf - dir | gzip > dir.tar.gz
```

where we've told the `tar` command to send its output directly into the `gzip` command by specifying a hyphen as the target file name. Now, I want to delete the original directory:

```
$ rm -rf dir
```

To reverse the operation, I just run the command line:

```
$ gzip -c dir.tar.gz | tar xfp -
```

making the `gzip` command send its output into the `tar` program. This unpacks the archive and recreates the original tree.

I used to type these commands, and as you can see, they are short and quick to use. However, one day I realized that I was often compressing files and decided to create a command file to do the task for me.

Designing a Script

I could just copy the store-and-compress one-liner into a file and would have a new command. However, that command would only work on the directory name that I entered in the file. I want to be able to select the source directory when I run the command, so the command file would need to take an argument. I want be able to say something like this:

```
$ tpack dir
```

Also, is it a good idea to delete the data in the script? What happens if things go wrong? What happens when I've given this new command a large directory to pack, and then leave the terminal? If the `tar` and `gzip` fail for some reason, am I happy that the files will be deleted? Of course, the answer is no. Too many bad things can happen, and data is precious.

One alternative is to rename the directory that has been packed up, so that when I arrive later, having forgotten I packed it, I can identify directory trees that can be deleted.

Here's the start of the script:

```
#!/bin/sh
```

The first line tells the system that it is a shell script, and that it should use the command `/bin/sh` to interpret and execute the script. In some circumstances, you don't need this initial line, but it's a good idea to get into the habit of adding it to a file. Continuing on:

```
tar cf - $1 | gzip > $1.tar.gz
mv $1 $1.dele
```

Now if you look back at the original commands, you'll see that the packing and compression text here is similar to the lines I typed before. The only difference is I've replaced `dir` with `$1`. Shell scripts work by "text replacement"; when the command is run, the `$1` is replaced by the contents of the first argument to the script. If the script had two arguments, then you could access the contents of the second argument by typing `$2`.

The `$1`, `$2`, etc. are a special case of *shell variables*, names in the shell that are replaced by the text they contain. You can always type something like this:

```
$ s=/home/user/dir/down
```

Note: there should be no spaces around the equals sign. We use it by saying

```
$ cp onefile $s/onefile
```

and copy the file to the expanded target directory. This trick can be useful if you are copying several files and don't want to do much typing. Incidentally, if you are a `csh` user, you have to set the variable, as follows (note the spaces):

```
$ set s = /home/user/dir/down
```

Making It Better

Our script will now work, but are there any snags? What happens if we call this script by accident with no arguments? When there are no arguments, the `$1` is replaced by an empty string and the command will still run. Something should be done to make the script more robust. We could put in a test and cause the script to die in these circumstances. However, I will side-step the problem by observing that at some point I am bound to want to run this command on several directories, so ideally, I want to be able to give it several arguments on the command line:

```
$ tpack joe fred jim
```

To handle several arguments, I need to place my commands into a loop and step through the arguments one at a time.

Here's the revised script:

```
#!/bin/sh
for dname in $*
do
     tar cf - $dname | gzip > $dname.tar.gz
     mv $dname $dname.dele
done
```

Again, the pattern of the commands is identical to the text we had before, except that `$1` has been replaced by `$dname`. The shell performs exactly the same replacement operation, but in this script, the directory name will be taken from the `dname` variable. In the script, `dname` is the loop variable, and the `for` statement tells the shell to repetitively execute the commands between `do` and `done`, setting `$dname` to the next value taken from the list that follows the word `in`.

Sure, there's no list of values after `in`, it's another magic internal shell variable: `$*`. When the script is run, `$*` is replaced by all the arguments to the command, creating a list of values that is used by the `for` statement. If we call our script with

```
$ tpack joe jim jack
```

then `$*` is replaced by `joe jim jack` when the `for` loop is initiated. The loop starts with the `dname` variable being set to `joe`, then to `jim` and, finally, to `jack`. The list after `in` is now empty and the loop terminates.

When the script is called with no arguments, the list after `in` starts off being empty and the commands inside the loop are not executed. As a result, our new script is safer when called with no arguments.

Creating a Command

We've created the command in the editor, what next? Well, I've assumed that the new command will be called `tpack`, so we will need to write our text into a file called `tpack`. We need to turn this file into a command that the shell can execute, and do this by setting the execute bit:

```
$ chmod +x tpack
```

We can now execute the new command by typing `tpack` into the shell. Incidentally, if you still get a "Command not found" error, try typing `./tpack` (read on to understand why).

If you are planning to run this command from your home directory compressing and saving directories, then you may be happy with what we have constructed. However, this is often not the case. Directories will be littered all over the system and we need some way of telling the shell where to look for the `tpack` command. I, along with many other people, have a private `bin` directory that lives under my home directory. I will want to place the `tpack` command into my `bin` and tell my shell to look in that directory when I type a command.

The shell uses an environment variable, `PATH`, to contain a colon-separated list of directories that are to be searched whenever you type a command. You can see the current value of `PATH` by typing

```
$ echo $PATH
:/usr/bin:/usr/ucb
```

In this example, the shell will search the current directory (shown by the initial empty string before the first colon), then `/usr/bin/` and, finally, `/usr/ucb`. You may find that the `PATH` variable already contains a private `bin` directory, and you have a thoughtful systems administrator to thank. If not, then we can add to this list by typing

```
$ PATH=:/home/pc/bin:/usr/bin:/usr/ucb
```

This uses the normal shell variable definition statement. You should replace `/home/pc/bin` by the path to your own private `bin` directory. There's one further piece of magic. Setting a variable like this only sets it in the current shell. If you want to pass the new string into all the commands you use, which you generally do, then you need to say

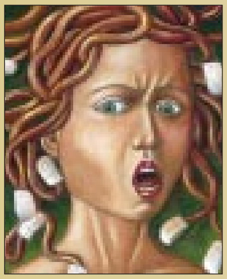
```
$ export PATH
```

Having done all that, you can now happily type `tpack` from any directory in the file system and your shell will locate it in your private `bin` directory.

Incidentally, things are a little different in `csh` and its derivatives. The `csh` shell's search path is controlled by the `PATH` variable and you set this using a list enclosed in round brackets, where the directory names are separated by spaces:

```
set path=(. /home/pc/bin /usr/bin /usr/ucb)
```

Here the dot character is used to indicate that the shell should search in the current directory, then my private `bin`, then `/user/bin` and, finally, `/usr/ucb`. The `PATH` variable possesses some special properties. Changing it will automatically change the environment setting for `PATH`, exporting the change to other programs.



The shell uses an environment variable, PATH, to contain a colon-separated list of directories that are to be searched whenever you type a command.

You don't want to have to type this information into your shell every time, so you need to place the setup line (or lines) into your shell startup file. The precise name of this file depends on the shell you're using. If you are unsure, consult a local guru.

More Information

You can find the Halloween documents at `http://www.opensource.org`, and if you haven't seen them they are well worth a visit. Eric S. Raymond's "The Cathedral and the Bazaar" can be found at `http://www.tuxedo.org/~esr/writings`. If you are interested in news from the open-source movement, then a visit to `http://www.slashdot.org` may be illuminating. My aforementioned article, "The Trouble with Windows," can be found at `http://sun.expert.com/C2/1997`–it's a PDF file so you'll need a copy of Adobe Systems Inc. Acrobat.

***Peter Collinson** runs his own UNIX consultancy, dedicated to earning enough money to allow him to pursue his own interests: doing whatever, whenever, wherever... He writes, teaches, consults and programs using Solaris running on a SPARCstation 2. Email:* `pc@cpg.com`.