But is it UNIX?®[*]
or why I hate OpenOffice

Greg Lehey
LEMIS (SA) Pty Ltd
grog@lemis.com

3 September 2003

Introduction

UNIX has been around for over a third of a century, well more than half the history of electronic computing. During that time the man/machine relationship has changed markedly. Newer, "user-friendly" systems have appeared. Many people believe that they are the way of the future, that UNIX is antiquated and hard to use, and that the UNIX user interface should be "fixed".

There is currently a trend to introduce Microsoft-like desktop software to UNIX. How does this concept blend with the UNIX philosophy? How well does it advantage of the system services? How easy is it to use?

This paper looks at the following aspects of "desktop" software and traditional UNIX software:

- Communicating with computers: use of keyboard, mouse and menus.
- The UNIX way of doing things.
- Tools used for communicating with computers.
- Menus.
- Icons.
- File formats.
- Solving the problem.

[*]UNIX is a trade mark of the Open Group. The UNIX copyrights belong to Novell Inc. The UNIX marketing rights belong to SCO. The UNIX heritage belongs to the community. All statements believed correct at the time of going to press.

Most examples will come from the area in which I have the most experience, document preparation.

For the sake of convenience, this paper refers to this "desktop" software as *desktop GUI*. The term applies to products from Microsoft and Apple as well as free software like *OpenOffice*.

Communicating

Conceptually, the user interface is a form of communication: to control a computer, you must communicate with it. Communications between humans and computers are modeled on various forms of communications between humans and humans. It's instructive to look at the evolution of communication between humans. In this paper I refer to least the following four levels of communication:

1. The most primitive form of communications, which predate language, are gestures and grimaces, which can include physical violence. It's true that people use these methods with computers as well, though few of them have any positive outcome. Some have been adapted, however: "point and click". This level has the advantage of being easy to learn.

2. The real breakthrough is speech, of course. Many consider the gift of speech to be the prime distinction between humans and animals. Speech didn't happen all at once, of course; initially it would almost certainly have been limited to a few concepts used to express concrete objects. In many ways, this is the language used by more primitive command line interfaces. The word *command* limits the scope of the language, of course: the communication we're talking about here is the human telling the computer what to do. In this situation, primitive humans might say "give food" or "build shelter": a verb (imperative) and a noun. Primitive computer command lines look similar: *edit document* or *remove file*.

3. Languages evolve. Human languages developed other parts of speech, notably adjectives and adverbs. That may seem like a small difference, but it's important. The previous level, consisting of verbs and nouns, was more accurate and less difficult than sign language, but it didn't really do much that couldn't be done with sign language. With adverbs and adjectives people were able to leave sign language behind and express nuances that sign language couldn't.

   Computer command languages have corresponding nuances as well. They're usually called options. You might compare the UNIX command *ls* with a command "show". Similarly, *ls -l* might correspond to "show in detail".

4. So far, language has only been used for simple operations. What about expressing more abstract concepts? Imagine a primitive human telling somebody what to expect in the forest: "If you see an elephant, climb up a big tree and hide". Or maybe he's standing in front of a large pile of debris which has blown in front of his cave. He wants to give instructions:

> "Take a little bit of this mess and throw it over the cliff, then come back and repeat until they're all gone". Both these activities are conditional, and the second is repetitive. In computer terms, they're programs.

None of these comparisons are new. All four levels of communication have been known for decades. Paradoxically, the most primitive level was one of the last to appear: initially the hardware requirements were too great.

The UNIX way

UNIX was successful for a number of reasons, but very little of it was original. One of the main reasons for its success were the close integration of a relatively small number of concepts which, when used together, proved to be a powerful environment in which to build solutions which were frequently elegant and efficient. To quote the committee which awarded Thompson and Ritchie the Turing award in 1983,

> The success of the UNIX system stems from its tasteful selection of a few key ideas and their elegant implementation. The model of the UNIX system has led a generation of software designers to new ways of thinking about programming. The genius of the UNIX system is its framework, which enables programmers to stand on the work of others.

Elsewhere,[1] the UNIX philosophy has been defined as:

- Write programs that do one thing and do it well.
- Write programs that work together.
- Write programs that handle text streams, because that is the universal interface.

The use of a few, simple file formats encouraged interchange of data between individual applications. In this respect it resembles a super-language for communication with the computer.

These goals are conceptually not closely related to the goal of "get the job done". UNIX appeared at a time when most users were either computer experts or had access to a computer expert when they ran into trouble. That changed greatly with the introduction of "personal" computers, which might be used by a single inexperienced person without access to help. Beginners had trouble both with the concepts and with operating the machines. More importantly, though, they weren't interested in doing things the right way. They just wanted to get them

[1]This statement is widespread, but it's not clear where it originated.

done. A large majority of users could not type effectively, so the keyboard was a problem. Many systems, UNIX included, had a large number of commands with often arcane names and usages. Users had difficulty remembering how to do things.

One reaction of the industry was to introduce “easy to use” software to make life easier for beginners. Some of the central concepts in this approach were menus and mice: the computer gave the user a selection of what he might want to use, and he could select it with the mouse. If the UNIX approach is a super-language, this approach looks more like sign language, level 1 of communication.

Other UNIX features

In addition to the “UNIX philosophy”, there are other important concepts about UNIX:

- UNIX is a multi-user system. Initially, “desktop” systems were intended for use by only one person at a time. While most UNIX desktops are also used by only one person at a time, the underlying implementation provides a number of standardized concepts which most software in the Microsoft space implement individually.
- UNIX has a standardized, highly structured file system layout. Although “desktop” software such as Microsoft also has such a structure, it is not used or enforced to anything like the same extent. In particular, each user has a “home directory” where most of his files reside.

The personal computer approach has been refined over the last twenty years to a point where all such software has relatively similar “look and feel”. It currently emphasizes a large number of selection buttons and much use of the mouse, even in cases where the keyboard would be a better choice. It frequently uses complicated file formats. Even in the case of standardized formats, data interchangeability is limited by the complexity of the format. The complexity of the software frequently gives rise to substandard reliability, security and efficiency.

The tools of communication

In the course of computer history, a number of different methods have been used for humans to communicate with computers:

- Initially, access was non-interactive, for example punched cards or paper tape.
- Round the time UNIX was introduced, it became practicable to use teletypes for interactive access to machines, and UNIX made great use of it. The output was on paper, so interaction tended to be linear.

- Not long afterwards, the first CRT-based terminals became available. They were character based serial devices which could seldom transfer more than 2 kB/s. They were often given the name *glass tty*, and much software treated them in exactly the same way as a teletype. Nevertheless, some "full-screen" software used this interface to create menus. The user generally navigated the menu with the cursor keys.

- In the 1980s, bit-mapped graphics became generally available. Such systems were usually equipped with a mouse. Initially available systems were proprietary, but in the late 1980s the X Window System became more generally available.

The advent of the X window system was too late to have any significant influence on the "desktop" market, which had developed in some isolation from UNIX, despite the fact that Microsoft marketed UNIX (under the trade name XENIX) in the 1980s. Microsoft's initial software development paralleled that of UNIX, with the exception that by the time of the introduction of the IBM PC, (primitive) bit-mapped graphics were available. Microsoft started using the bit-mapped interface in the mid- to late 1980s.

At this point, there was little difference in the way Microsoft and UNIX approached the issue of interactive I/O. The most notable difference was that UNIX had a powerful command language, the shell. In terms of the hierarchy described above, it matched the fourth level, with programming concepts. Microsoft had a command language which embraced similar concepts, but it was far less powerful and matched the third level.

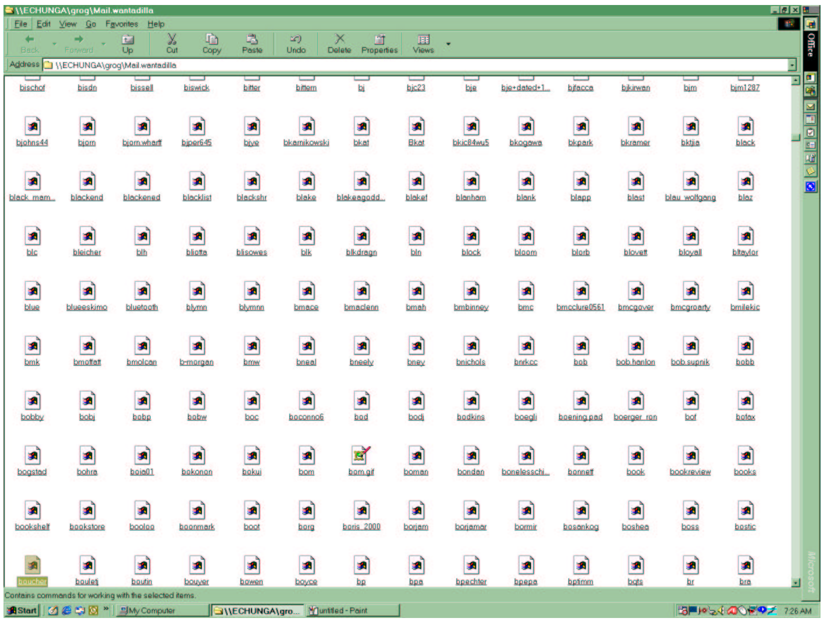
Menus

By the early 1990s, all systems, including UNIX, made significant use of menus. No text menu based applications made their way into UNIX, but the X Window system made significant use of them, though not to the same extent as the desktop GUI approach, where they are central to the user interface. For example, the basic access to directories and files is via a menu. This imposes significant limitations on the size of directories: it is impractical to have menus with more than 100 entries, and even this many entries make things complicated. Although UNIX also has some problems with large directories, the typical UNIX directory is not constrained by the graphical representation and can be much larger. In some cases, this is necessary: some applications naturally map a specific entity to a file or directory. Email is a good example: a common way to archive Email is in files which match the name of the sender. I have something over 8,000 folders (mail files) in my *Mail* directory.

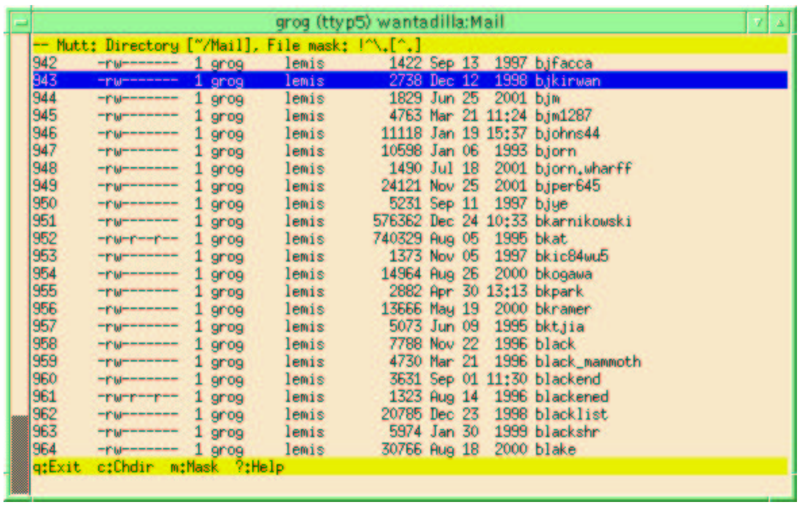
> For some reason, Microsoft has phased out the word *directory* in favour of the word *folder*. This is confusing and unnecessary: in Microsoft as elsewhere, the term *folder* is used for a collection of mail messages, which may be stored in individual files in their own directory, in a single file, or

in a database. Unfortunately, some free desktop GUI software is following this example.

With Microsoft's *Outlook*, looking for a folder in such a directory can look like this:

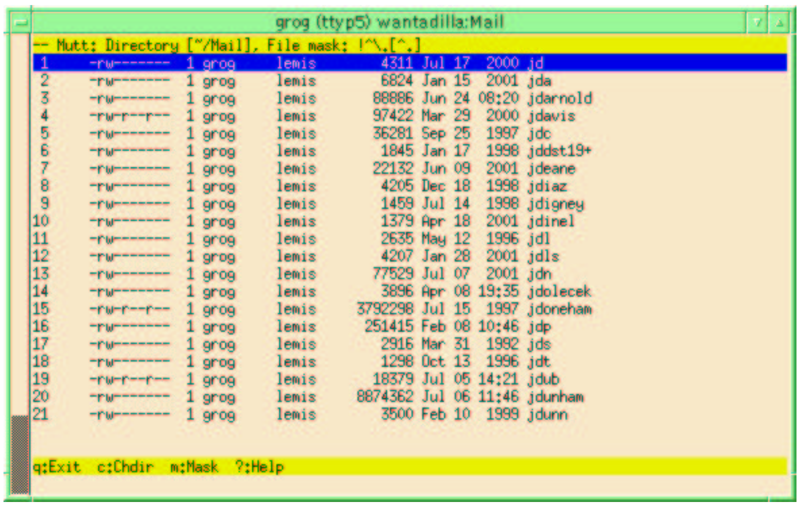

\\ECHUNGA\grog\Mail.wantadilla
File Edit View Go Favorites Help
Back Forward Up Cut Copy Paste Undo Delete Properties Views
Address \\ECHUNGA\grog\Mail.wantadilla
bischof bisdn bissell biswick bitter bittern bj bjc23 bje bje+dated+1... bjfacca bjkirwan bjm bjm1287
bjohns44 bjorn bjorn.wharff bjper645 bjye bkarnikowski bkat Bkat bkic84wu5 bkogawa bkpark bkramer bktjia black
black_mam... blackend blackened blacklist blackshr blake blakeagodd... blakef blanham blank blapp blast blau_wolfgang blaz
blc bleicher blh bliotta blisowes blk blkdragn bln block bloom blorb blovett bloyall bltaylor
blue blueeskimo bluetooth blymn blymnn bmace bmaclenn bmah bmbinney bmc bmcclure0561 bmcgover bmcgroarty bmilekic
bmk bmoffatt bmolcan b-morgan bmw bneal bneely bney bnichols bnrkcc bob bob.hanlon bob.supnik bobb
bobby bobj bobp bobw boc boconno6 bod bodj bodkins boegli boening.pad boerger_ron bof bofax
bogstad bohra boia01 bokonon bokui bom bom.gif boman bondan bonelesschi... bonnett book bookreview books
bookshelf bookstore booloo boonmark boot borg boris_2000 borjam borjamar bormir bosankog boshea boss bostic
boucher boulet boutin bouyer bowen boyce bp bpa bpechter bpepa bptimm bqts br bra
Contains commands for working with the selected items.
Start My Computer \\ECHUNGA\gro... untitled - Paint 7:26 AM
Office
Microsoft

This is one page of nearly 100. Clearly it's very difficult to search this list by paging through it or scrolling with the scroll bar. In many ways this applies to the corresponding UNIX list (from the *mutt* MUA):



```
grog (ttyp5) wantadilla:Mail
-- Mutt: Directory [~/Mail], File mask: !^\.[^.]
942    -rw-------  1 grog    lemis       1422 Sep 13  1997 bjfacca
943    -rw-------  1 grog    lemis       2738 Dec 12  1998 bjkirwan
944    -rw-------  1 grog    lemis       1829 Jun 25  2001 bjm
945    -rw-------  1 grog    lemis       4763 Mar 21 11:24 bjm1287
946    -rw-------  1 grog    lemis      11118 Jan 19 15:37 bjohns44
947    -rw-------  1 grog    lemis      10598 Jan 06  1993 bjorn
948    -rw-------  1 grog    lemis       1490 Jul 18  2001 bjorn.wharff
949    -rw-------  1 grog    lemis      24121 Nov 25  2001 bjper645
950    -rw-------  1 grog    lemis       5231 Sep 11  1997 bjye
951    -rw-------  1 grog    lemis     576362 Dec 24 10:33 bkarnikowski
952    -rw-r--r--  1 grog    lemis     740329 Aug 05  1995 bkat
953    -rw-------  1 grog    lemis       1373 Nov 05  1997 bkic84wu5
954    -rw-------  1 grog    lemis      14964 Aug 26  2000 bkogawa
955    -rw-------  1 grog    lemis       2882 Apr 30 13:13 bkpark
956    -rw-------  1 grog    lemis      13666 May 19  2000 bkramer
957    -rw-------  1 grog    lemis       5073 Jun 09  1995 bktjia
958    -rw-------  1 grog    lemis       7788 Nov 22  1996 black
959    -rw-------  1 grog    lemis       4730 Mar 21  1996 black_mammoth
960    -rw-------  1 grog    lemis       3631 Sep 01 11:30 blackend
961    -rw-r--r--  1 grog    lemis       1323 Aug 14  1996 blackened
962    -rw-------  1 grog    lemis      20785 Dec 23  1998 blacklist
963    -rw-------  1 grog    lemis       5974 Jan 30  1999 blackshr
964    -rw-------  1 grog    lemis      30766 Aug 18  2000 blake
q:Exit  c:Chdir  m:Mask  ?:Help
```

The difference in the way the files are displayed is not important: you can display Microsoft folders in textual form as well. The problem is finding the correct needle in your haystack. The traditional desktop GUI approach has been to scroll the scroll bar and then click on the appropriate field, all with the mouse. This is not easy: the scroll bar has become very small, and it's difficult to select it. Once you have, a small movement can scroll more than a page. Even when you master this problem, maybe by using the `PgUp` and `PgDn` keys, you need to recognize the correct file name. It's surprisingly easy to miss it, especially with the icon layout. It's a lot easier to use the keyboard. A technique used in UNIX is the *incremental search*: the user types in the initial characters of the text being searched, and the cursor is positioned on the first text which matches. After typing, say, `jd`, the display becomes:



```
-- Mutt: Directory [~/Mail], File mask: !^\.[^.]
 1     -rw-------  1 grog     lemis        4311 Jul 17  2000 jd
 2     -rw-------  1 grog     lemis        6824 Jan 15  2001 jda
 3     -rw-------  1 grog     lemis       88886 Jun 24 08:20 jdarnold
 4     -rw-r--r--  1 grog     lemis       97422 Mar 29  2000 jdavis
 5     -rw-------  1 grog     lemis       36281 Sep 25  1997 jdc
 6     -rw-------  1 grog     lemis        1845 Jan 17  1998 jddst19+
 7     -rw-------  1 grog     lemis       22132 Jun 09  2001 jdeane
 8     -rw-------  1 grog     lemis        4205 Dec 18  1998 jdiaz
 9     -rw-------  1 grog     lemis        1459 Jul 14  1998 jdigney
10     -rw-------  1 grog     lemis        1379 Apr 18  2001 jdinel
11     -rw-------  1 grog     lemis        2635 May 12  1996 jdl
12     -rw-------  1 grog     lemis        4207 Jan 28  2001 jdls
13     -rw-------  1 grog     lemis       77529 Jul 07  2001 jdn
14     -rw-------  1 grog     lemis        3896 Apr 08 19:35 jdolecek
15     -rw-r--r--  1 grog     lemis     3792298 Jul 15  1997 jdoneham
16     -rw-------  1 grog     lemis      251415 Feb 08 10:46 jdp
17     -rw-------  1 grog     lemis        2916 Mar 31  1992 jds
18     -rw-------  1 grog     lemis        1298 Oct 13  1996 jdt
19     -rw-r--r--  1 grog     lemis       18379 Jul 05 14:21 jdub
20     -rw-------  1 grog     lemis     8874362 Jul 06 11:46 jdunham
21     -rw-------  1 grog     lemis        3500 Feb 10  1999 jdunn


q:Exit  c:Chdir  m:Mask  ?:Help
```

This method is much easier than the "user friendly" desktop GUI method, and some desktop GUI software, including Microsoft Outlook, has adopted it, although its use is not widely known: apparently it differs too much from the traditional approach.

Menu trees

An alternative solution to the size of menus is to arrange them in a hierarchy. This works well for programs. For example, on my laptop I have the following number of executables:

Directory	Number of executables
/bin	40
/usr/bin	406
/sbin	104
/usr/sbin	229
/usr/local/bin	630
/usr/local/sbin	35
/usr/X11R6/bin	353
Total	1797

It's almost impossible to remember the names and purposes 1800 different programs, and many are of only limited utility. A menu system which can present the most useful ones in a clear and understandable way can be a great advantage. The question is how to present them. It's clearly impractical to display a single menu with 1,800 entries: it's more difficult to navigate than remembering the names. There are two possible solutions:

- Present submenus with related programs.
- Ignore most of the programs, notably the more specialized ones.

Current Microsoft implementations use both of these techniques. The normal method of starting a program selects at least one and possibly multiple nested submenus. Even so, a number of programs cannot be found via the menus, and others are placed in non-intuitive locations in the menu tree. These programs can be started from a special one-line window, selected with a mouse click, into which the user types the name of the program.

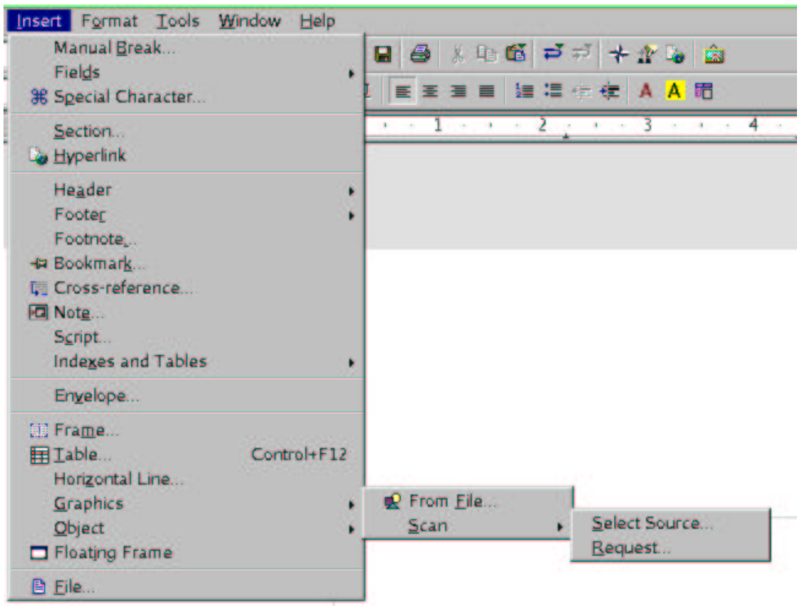
For some people, this is a useful compromise. It makes life easier for people who can't type well, and who use only a small subset of what the system has to offer. The disadvantages still exist, but many perceive them to be a reasonable price to pay for the convenience of finding a program easily and not having to type much.

For other people, particularly experienced computer users, this approach is painful. Instead of simply typing in the name of a program, you need to:

1. Find the mouse.
2. Select the "start" icon, often at bottom left of the screen.
3. Select a submenu from the menu.
4. Possibly repeat, selecting a further submenu from the submenu.
5. Select the final program.
6. If the program requires arguments, supply them by whatever means the program provides.

This last point shows a further problem with the approach: if it's difficult to list all programs in a menu, it's clearly impossible to include every possible option for every program. The result? Programs are written not to require options. This is a workaround, not a solution.

The alternative way to present options is, of course, a menu or menu tree. For example, to tell *OpenOffice* to scan a document, you go through the following tree:



The normal way to get to this menu is to select the `Insert` tab at the top of the screen, then pass the mouse over the `Graphics` tab, which causes the first submenu to appear. Selecting `Scan` then causes the next menu to appear. With practice, this does not take more than about two seconds, not much more than the time it takes a practiced typist to type in a command.

There are better alternatives, though, as indicated by the underlined letters on the menus: you can invoke the same submenu with the keyboard sequence `Alt-IGS`. This is much closer in concept to the command line approach. The difference is that the character sequence `scan` is relatively easy to remember; `Alt-IGS` is not. In addition, there is no record of the keystrokes which have been pressed.

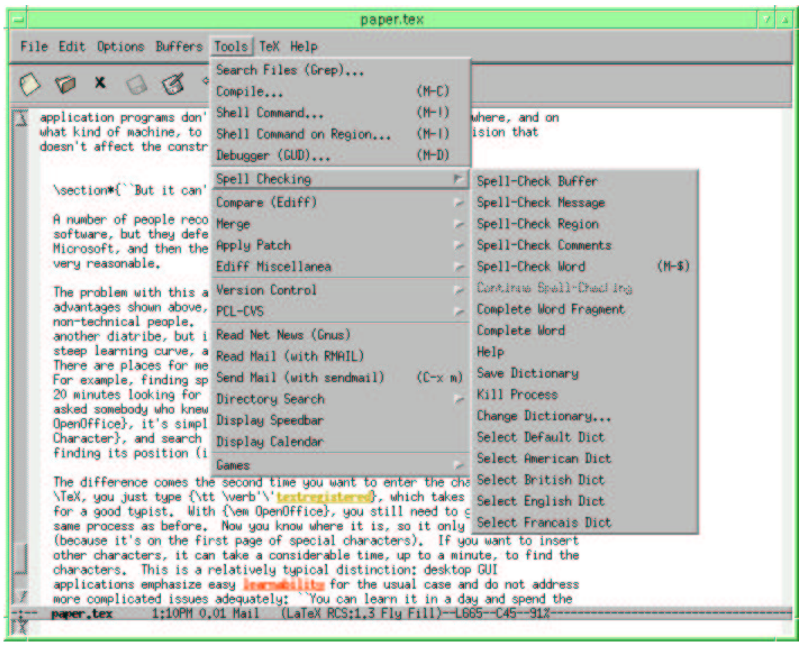
What's wrong with the UNIX way?

UNIX isn't perfect either: UNIX desktop software is not ready for use by non-technical people. The problems with UNIX software are sufficient for another diatribe, but in summary it's not really "user-friendly". There's a steep learning curve, and a lot of things require deep magic to use at all. There are places for menus and other desktop GUI features in UNIX software. For example, finding special characters for use in TEX is difficult. I don't normally use TEX, and while preparing this document I spent 20 minutes looking for the character ®. I finally gave up and asked somebody who knew (it's `\textregistered`). In *OpenOffice*, it's simple: you Select `Insert`, then `Special Character`, and search the list for the character. All in all, including finding its position (it's in the first page, ISO 8859-1), less than a minute.

The difference comes the second time you want to enter the character. In

TEX, you just type `\textregistered`, which takes about 2 seconds for a good typist. With *OpenOffice*, you still need to go through the same process as before. Now you know where it is, and because it's on the first page of special characters, it only takes 10 seconds. If you want to insert characters located on other pages, it can take a considerably longer time, up to a minute, for every instance of the character. This is a relatively typical distinction: desktop GUI applications emphasize easy learnability for the usual case and do not address more complicated issues adequately: "You can learn it in a day and spend the rest of your life paying for it".

Clearly there should be something better than either approach. I take the position that the UNIX approach is more expandable than the desktop GUI approach. But concentrating programmer effort desktop GUI applications distracts from the task of finding a better approach. Such better approaches exist: the *Emacs* editor, for example, has the reputation of being very difficult to use, partially helped by its author's insistence on mapping the `Backspace` key to the help function. Nevertheless, *Emacs* is as easy to use as any editor: it supplies the same menu functionality as most desktop GUI software, but it doesn't *require* the use of the menus:



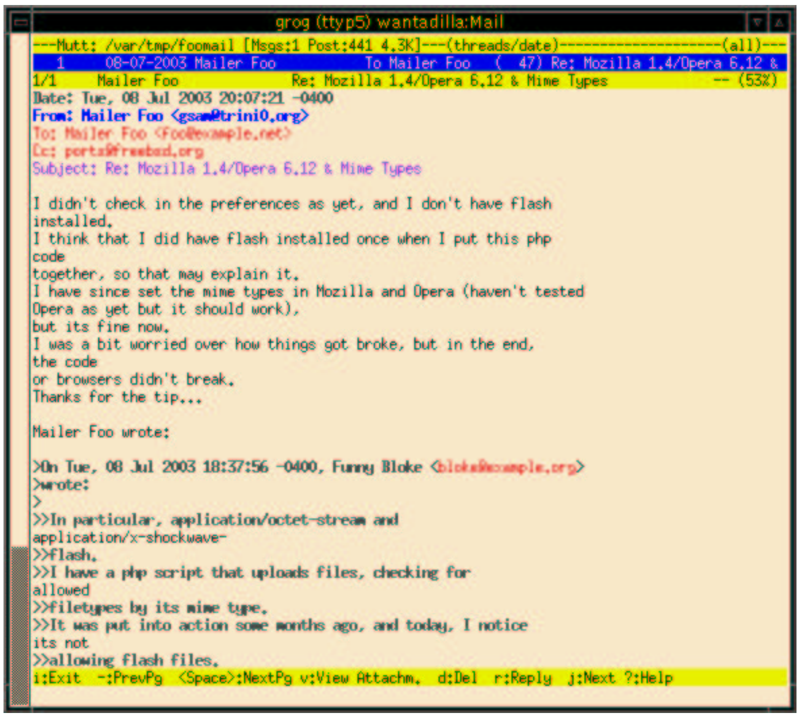
```
application programs don'                               where, and on
what kind of machine, to                                ision that
doesn't affect the constr

  \section*{``But it can'

  A number of people reco
  software, but they defe
  Microsoft, and then the
  very reasonable.

  The problem with this a
  advantages shown above,
  non-technical people.
  another diatribe, but i
  steep learning curve, a
  There are places for me
  For example, finding sp
  20 minutes looking for
  asked somebody who knew
  OpenOffice}, it's simpl
  Character}, and search
  finding its position (i

  The difference comes the second time you want to enter the cha
  \TeX, you just type {\tt \verb'\'textregistered}, which takes
  for a good typist.  With {\em OpenOffice}, you still need to g
  same process as before.  Now you know where it is, so it only
  (because it's on the first page of special characters).  If you want to insert
  other characters, it can take a considerable time, up to a minute, to find the
  characters.  This is a relatively typical distinction: desktop GUI
  applications emphasize easy learnability for the usual case and do not address
  more complicated issues adequately: ``You can learn it in a day and spend the
-:-- paper.tex      1:10PM 0.01 Mail   (LaTeX RCS:1.3 Fly Fill)--L665--C45--91%----
```

Where key bindings exist for the functions, the key presses are listed on the menu, in the same manner as with *OpenOffice*. The notation `M-C` means the *Meta* (*Alt*) key with the letter `C`. The difference from *OpenOffice* is that *all* functions can be bound to keys.

Emacs is by no means perfect. It does a lot of things that desktop GUI applications do, but there aren't enough people working on it: the "fun" stuff is the desktop GUI. As a result, *Emacs* has some loose edges.

A bigger problem is that most desktop GUI users don't understand the power of an editor. They're happy with the level of text manipulation that *OpenOffice* and Microsoft's "Word" provide, or at least they don't go to any trouble to find something better.

The lack of editing capability hits you everywhere, but the most impressive example is Email. There's a strong correlation between the appearance of a mail message and the software with which it was written. Messages written with a desktop GUI are typically written in a single block at the top of the message, with any prior correspondence tacked on the end, frequently with extreme mutilation of the text:



```
grog (ttyp5) wantadilla:Mail
---Mutt: /var/tmp/foomail [Msgs:1 Post:441 4.3K]---(threads/date)--------------------(all)---
   1     08-07-2003 Mailer Foo              To Mailer Foo    (  47) Re: Mozilla 1.4/Opera 6.12 &
1/1      Mailer Foo               Re: Mozilla 1.4/Opera 6.12 & Mime Types                 -- (53%)
Date: Tue, 08 Jul 2003 20:07:21 -0400
From: Mailer Foo <gsam@trini0.org>
To: Mailer Foo <Foo@example.net>
Cc: ports@freebsd.org
Subject: Re: Mozilla 1.4/Opera 6.12 & Mime Types

I didn't check in the preferences as yet, and I don't have flash
installed.
I think that I did have flash installed once when I put this php
code
together, so that may explain it.
I have since set the mime types in Mozilla and Opera (haven't tested
Opera as yet but it should work),
but its fine now.
I was a bit worried over how things got broke, but in the end,
the code
or browsers didn't break.
Thanks for the tip...

Mailer Foo wrote:

>On Tue, 08 Jul 2003 18:37:56 -0400, Funny Bloke <bloke@example.org>
>wrote:
>
>>In particular, application/octet-stream and
application/x-shockwave-
>>flash.
>>I have a php script that uploads files, checking for
allowed
>>filetypes by its mime type.
>>It was put into action some months ago, and today, I notice
its not
>>allowing flash files.
i:Exit  -:PrevPg  <Space>:NextPg v:View Attachm.  d:Del  r:Reply  j:Next ?:Help
```

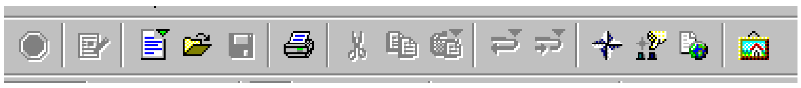
In this example, written with Mozilla/5.0, the lengths of the lines are uneven, the text of the original messages is attached below the text, and the quote levels vary from one line to the next. It also contains a lot of irrelevant text not shown in this view of the message. All this is the result of inadequate mail software and the lack of an editor. It's reasonable to assume that the originator of this message didn't intend it to look this bad, but that it was too difficult to change.

Icons

Most desktop GUIs use some form of *icon*, a small image that represents a certain function that can be performed by selecting it with the mouse, or sometimes with the keyboard. They also correspond to a more primitive method of writing than current western alphabets (pictograms instead of hieroglyphs).

They're very useful for small children and other people who can't read, but they have a number of significant disadvantages:

- They need to be learnt before they can be understood. The following icons come from *OpenOffice*, a number of them unique to *OpenOffice*:

  It's possible to learn what they mean by passing the cursor over the icon, whereupon a possibly descriptive text appears. In time you can learn what they mean, but you need to learn it: if the descriptive text were present, there would be nothing left to learn, except possibly how to read. I maintain that it's better to learn a generalized skill such as reading rather than the specialized skill of learning icons for a particular application.

- They don't scale according to the size of the display. They have the same size in pixels on a 640x480 display (where they take up too much of the surface area) as they do on a 2048x1536 screen (where they are unrecognizable).

- Using them requires taking your hands off the keyboard.

- They're difficult to position on, especially on a high resolution display.

File formats

UNIX file formats are nearly all in plain text; this greatly helps interoperability. The text of this paper was originally written for the *groff* formatter. Part of the source text looked like this:

```
Elsewhere, the UNIX philosophy has been defined as:
.Ls B
.LI
Write programs that do one thing and do it well.
.LI
Write programs that work together
.LI
Write programs that handle text streams, because that is the
universal interface.
.Le
```

The proceedings of the AUUG conference are formatted from LaTeX sources, so I changed the format accordingly. The same passage now looks like this:

```
    Elsewhere, the UNIX philosophy has been defined as:

    \begin{itemize}
```

```
\item
  Write programs that do one thing and do it well.
\item
  Write programs that work together
\item
  Write programs that handle text streams, because that is the
  universal interface.
\end{itemize}
```

This conversion can be made with a text editor. It is greatly assisted by having a programmable editor such as GNU *Emacs*. Both formats have the great advantage that other utilities, such as *grep*, *diff* and *wc*, can handle them directly. For example, the second line of text above was missing a full stop. After fixing it, *diff* shows:

```
--- paper.tex 2003/07/08 03:39:38 1.3
+++ paper.tex 2003/07/08 03:39:45
@@ -130,7 +130,7 @@
   \item
     Write programs that do one thing and do it well.
   \item
-    Write programs that work together
+    Write programs that work together.
   \item
     Write programs that handle text streams, because that is the
     universal interface.
```

This makes the changes immediately visible. There appears to be no way to perform a corresponding function with desktop GUI text processors.

The proponents of *OpenOffice* point out that *OpenOffice* also uses an "open format", XML. That is correct, but it misses a lot of the richness of the UNIX environment. In preparing this paper, I also imported this text into *OpenOffice*. Here's what I had to do:

- Highlight the text in an *Emacs* window and copy it into *OpenOffice* with mouse button 2.
- Remove the markup manually: *OpenOffice* does not have advanced editing facilities. Even simple functions like "delete to end of line" appear to be missing. This task was further complicated by the extreme proportional fonts that *OpenOffice* uses: the spaces are only about the width of the characters.
- Search for the tool which adds bullet points. It's available via the icon `Format-Numbering/Bullets`.
- For some reason, the first line came out underlined. Select it, then find the U icon to turn off underlining.

- Press `ctrl-S` to save the document. A menu appears showing a quarter of the home directory (not the current working directory), ordered with directories first. At the bottom is a window for the file name. To save in the current directory, I have to enter the entire pathname, helped somewhat by inbuilt file name completion.

- The file is saved with a different name: instead of *sampletext*, it is called *sampletext.sxw*.

- This file is 5045 bytes long. It is in some binary format. *file(1)* states:

```
$ file sampletext.sxw
sampletext.sxw: Zip archive data, at least v2.0 to extract
```

- *zip -l* tells us:

```
$ zip -l sampletext.sxw
Archive:  sampletext.sxw
  Length     Date   Time    Name
 --------    ----   ----    ----
     5582  07-08-03 04:10   content.xml
     5147  07-08-03 04:10   styles.xml
     1119  07-08-03 04:10   meta.xml
     6183  07-08-03 04:10   settings.xml
      752  07-08-03 04:10   META-INF/manifest.xml
 --------                   -------
    18783                   5 files
```

It's a reasonable expectation that the text is in *content.xml*. It consists of exactly two lines, the second unterminated and 5610 characters long. Clearly *diff* is not going to be of much use with it. Most of it is XML markup. The *Emacs* `fill-paragraph` macro makes it marginally legible: there are 118 lines, of which the original text is:

```
<text:p>text:style-name="P1">Elsewhere, the UNIX philosophy
has been defined as:</text:p><text:p
text:style-name="P2"/><text:unordered-list
text:style-name="L1"><text:list-item><text:p
text:style-name="P3"><text:s/>Write programs that do one
thing and do it
well.</text:p></text:list-item><text:list-item><text:p
text:style-name="P3"><text:s/>Write programs that work
together.</text:p></text:list-item><text:list-item><text:p
text:style-name="P3"><text:s/>Write programs that handle
text streams, because that is the universal
interface.</text:p></text:list-item></text:unordered-list>
<text:p text:style-name="Standard"/><text:p
text:style-name="Standard">Add some text here.</text:p>
```

Clearly this text is not intended to be read by humans.

This exercise may seem a little silly. Certainly it would be foolhardy to attempt to edit this document with the traditional UNIX tools. It's interesting to look at these steps for other reasons, though:

- *OpenOffice* does understand some of the X conventions, though there appear to be bugs in the implementation. For example, it can't paste from a different display.
- *OpenOffice* is not an editor. If you want an editor, you need to go elsewhere.
- *OpenOffice* does not have a freely exchangeable data format. Thus, after you've gone elsewhere for your editing, you have a data conversion issue.
- *OpenOffice* does not understand UNIX directories adequately. When saving a new document, it goes to the home directory, not the current directory..
- *OpenOffice* recognizes its files by the file type, but it adds an "extension" to the file name when saving. This is confusing for people who are used to free format file names.

It's important to distinguish here between bugs and implementation decisions. It's quite possible, for example, that the first point is a bug which can be fixed. The difficulty of data interchange is an implementation decision.

It's also important to understand here that this is not a particular criticism of *OpenOffice*. *OpenOffice* provides interesting examples of the problems which much desktop GUI software shares.

Multi this, multi that

UNIX is a multi-user system: since the very beginning, UNIX has run multiple processes, and as long back as anybody can remember, it has supported multiple users. By contrast, Microsoft has only recently taken the possibility of multiple users into consideration, and most of the support is for serial usage: one person uses the computer, stops, and allows somebody else to

The Microsoft paradigm grew up independently of networks; by contrast, UNIX has been involved with networking for more than 20 years. UNIX users routinely use networks as part of their work environment. The X window system is a networking protocol which allows physical and even geographical separation clients and servers. Most Microsoft-related software doesn't understand this. Here are a couple of examples of the limitations that this mind set causes:

- It is no longer possible to start multiple instances of most web browsers. Early browsers could do it, but complained about problems with the web page cache: they did not lock the cache, so any additional instances did not use the cache.

- More recent browsers, such as *mozilla*, recognize that an instance is running and require the user to choose another *profile*, really a reinvention of the UNIX user paradigm.
- *galeon* recognizes if an instance of *galeon* is already running. If it is, it simply opens another window on the display where the first instance is running, ignoring the value of the `DISPLAY` variable. This makes it impossible, for example, to have a second browser display on a laptop instead of on the home machine. It also makes it impossible to display windows on more than one display of a multi-headed machine.
- *OpenOffice* also attaches to a running instance, causing the same problems as with *galeon*.

"But it can't do any harm"

A number of people recognize the problems of the current crop of open source desktop software, but they defend its use: "It provides people with an alternative to Microsoft, and then they can learn what real software is about". This sounds very reasonable.

The problem with this approach is that this kind of software stifles alternatives. Only a certain amount of manpower is available for writing free software, and every programmer who writes software to emulate Microsoft is one fewer programmer available for writing a better alternative.

What needs to change

To fix the problems with desktop GUI software,

- The GUI needs to become better attuned to the needs of an experienced typist.
- Desktop GUI software needs to understand more of the UNIX environment.
- Desktop GUI software needs to interact better with the UNIX environment.

These problems are in increasing order of difficulty. Steps are already under way to solve the first problem. Almost no modern desktop GUI software relies entirely on the mouse for navigating the screen any more. Most offer "keyboard short cuts", in other words a variant of what UNIX has been doing all along. There is hope that this problem will be solved relatively quickly.

It should also be possible to teach desktop GUI software more about the UNIX environment. The problem here is not difficulty, it's a matter of recognizing the need. UNIX and Microsoft use some things differently, such as the keyboard and mouse. KDE has a solution to this problem: the user has the choice of a number of profiles, some similar to UNIX, others similar to Microsoft, and others again going the "KDE way".

The real problem is going to be the third issue: GUI software makes assumptions which are completely at odds with the UNIX model, for example the manner in which text is stored and presented. Some attempts have been made with programs like *LyX*, but they miss the real point: the file format should not be hidden. This is the basis on which programs like *grep*, *diff* and *cvs* are built: the user should be able to recognize the original text and know what it does.

This is not an easy change to sell. A whole generation of computer users have grown up with desktop GUI software, and most of them are happy with not knowing the details. They may not be happy with some aspects of what they have—most computer users complain about their software—but the vast majority is not prepared to change, let alone to something which looks primitive and difficult.

There will probably always be computer users who have little understanding of computers, and life should not be made more difficult for them. The real issue here is not that the desktop GUI interface exists, but that it is restrictive: it makes life difficult for more experienced users.

There's a vicious circle here: the interface won't change until people want it to change. People won't want it to change until they understand how things can be easier with the alternatives. Currently, all we can do is recognize that there's a problem; there's currently no solution, though it's clear where it would have to come from.