Volume 6, Number 2 December 1993 $2.50

MUUG Lines

Manitoba UNIX® User Group

Newsletter of the Manitoba UNIX® User Group

UNIX Commands Explained

Compiled by Ted Timar

(Submitted by Andrew Trauzzi)

Here is a condensed list of UNIX commands that you may have seen (and even used), but might not know what they stand for:

awk — "Aho Weinberger and Kernighan"
This language was named by its authors, Al Aho, Peter Weinberger and Brian Kernighan.

grep — "Global Regular Expression Print"
grep comes from the ed command to print all lines matching a certain pattern g/re/p where "re" is a "regular expression".

fgrep — "Fixed GREP".
fgrep searches for fixed strings only. The "f" does not stand for "fast" - in fact, "fgrep foobar *.c" is usually slower than "egrep foobar *.c" (Yes, this is kind of surprising. Try it.)

egrep — "Extended GREP"
egrep uses fancier regular expressions than grep. Many people use egrep all the time, since it has some more sophisticated internal algorithms than grep or fgrep, and is usually the fastest of the three programs.

cat — "CATenate"
catenate is an obscure word meaning "to connect in a series", which is what the "cat" command does to one or more files. Not to be confused with C/A/T, the Computer Aided Typesetter.

gecos — "General Electric Comprehensive Operating System"
When GE's large systems division was sold to Honeywell, Honeywell dropped the "E" from "GECOS". Unix's password file has a "pw_gecos" field. The name is a real holdover from the early days. Dennis Ritchie has reported: "Sometimes we sent printer output or batch jobs to the GCOS machine. The gcos field in the password file was a place to stash the info. for the $IDENT card. Not elegant."

nroff — "New ROFF", **troff**— "Typesetter new ROFF"
These are descendants of "roff", which was a re-implementation of the Multics "runoff" program (a program that you'd use to "run off" a good copy of a document).

biff — "BIFF"
This command, which turns on asynchronous mail notifica tion, was actually named after a dog at Berkeley. Biff was popular among the residents of Evans Hall, and was known for barking at the mailman, hence the name of the command.

rc (as in ".cshrc" or "/etc/rc") — "RunCom"
"rc" derives from "runcom", from the MIT CTSS system, ca. 1965. "There was a facility that would execute a bunch of commands stored in a file; it was called 'runcom' for 'run commands', and the file began to be called 'a runcom.' 'rc' in Unix is a fossil from that usage." Brian Kernighan & Dennis Ritchie mention that "rc" is also the name of the shell from the new Plan 9 operating system.

Perl — "Practical Extraction and Report Language"

Perl — "Pathologically Eclectic Rubbish Lister"
The Perl language is Larry Wall's highly popular freely-available completely portable text, process, and file manipulation tool that bridges the gap between shell and C programming (or between doing it on the command line and pulling your hair out).

This Month's Meeting

Meeting Location:
Our next meeting is scheduled for Tuesday, December 14, at 7:30 PM. The Annual MUUG Wine and Cheese Party will be held in the atrium of the St-Boniface Hospital Research Centre, just south of the hospital itself, at 351 Taché. You don't have to sign in at the security desk – just say you're attending the meeting of the Manitoba UNIX User Group. The atrium is on the main floor, and is easily seen from the entrance.

Meeting Agenda: Annual MUUG Open House!

Inside This Issue

RAMBLINGS

Happy Holidaze!

By Andrew Trauzzi

By the time you read this, the Christmas season should be in full swing! I'm sure there are many of you who, like me, wait until the 22nd or 23rd to do all of their shopping. I personally think it's part of the Christmas spirit, although I don't apply the same logic to newsletter editing. I attempt to have a final copy a week in advance of the mailing, but so far, it hasn't worked out that way. Here are a few regular columns I thought of for the newsletter — some I have implemented, and some are planned.

C++

Many of you have been asking for more technical articles such as X-Windows and Motif programming. Unfortunately, no one has volunteered to write a programming column, and I don't have the necessary experience in X-Window programming. Instead, I will be running a monthly column that will review C++ and object-oriented programming. The column's format will be mostly "canned" questions and answers from the various programming newsgroups. If any of you would like to ask a specific C++ question, I would be happy to answer it in the column. If you are unhappy with the column's content, please drop me a line

Beginning UNIX

I have been fielding many questions about UNIX that seem to be rather basic (to me, at least). Because of this, and the large number of MUUG members that are entirely new to UNIX, I have decided to publish a beginner's UNIX column that will outline basic (and not-so-basic) techniques you need in order to survive on-line. If you have any UNIX 'tips' that you find useful send them to me. The column wasn't ready for this month, but I've included an article on the background behind some strange UNIX acronyms and commands.

Internetworking

I know that the large majority of MONA members use the Internet frequently. I also know that a large number of you would like to know where to find scads of information and programs. This column will hopefully outline the history, techniques, and sites of the Internet. If any of you know of a site with something special (like hourly satellite weather maps), send me some mail outlining the site, what's there, the average transfer rate, and availability. I'm sure all the other MONA users would appreciate the information. Look for this column in upcoming issues.

The 1993-1994 Executive

President:	Bary Finch	(W) 934-2723
Vice-President:	Ramon Ayre	(W) 947-2669
Treasurer:	Rick Horocholyn	(W) 474-4533
Secretary:	Brad West	(W) 983-0336
Membership Sec.:	Greg Moeller	(H) 786-6132
Mailing List:	Roland Schneider	1-482-5173
Meeting Coordinator:	Roland Schneider	1-482-5173
Newsletter editor:	Andrew Trauzzi	(W) 986-6009
Publicity Director	Rory McLeod	488-5168
Past President	Susan Zuk	(W) 631-2530
Information:	Bary Finch	(W) 934-2723
		(FAX) 934-2620
(or)	Andrew Trauzzi	(W) 986-6009
		(FAX) 986-5966

Copyright Policy and Disclaimer

Advertising Rates

Quarter page	$50
Half page	$75
Full page	$100
Insert (1-4 pages)	$100

Above prices are per issue. The first ad is charged at the full price; each successive month is 1/2 price.

Ad copy must be submitted by the final copy deadline for an issue (usually 3 weeks prior to the monthly meeting) in a format acceptable to the editor. (Please make arrangements with editor beforehand.)

Internet E-mail: editor@muug.mb.ca

Group Information

The Manitoba UNIX User Group meets at 7:30 PM the second Tuesday of every month, except July and August. Meeting locations vary. The newsletter is mailed to all paid-up members one week prior to the meeting. Membership dues are $25 annually and are due as indicated by the renewal date on your newsletter's mailing label. Membership dues are accepted at any meeting, or by mail.

Manitoba UNIX User Group
P.O. Box 130, Saint-Boniface
Winnipeg, Manitoba R2H 3B4

Internet E-mail: membership@muug.mb.ca

PRESIDENT'S CORNER

It's That Time Again

By Bary Finch

Once again the festive season is upon us. We will all celebrate in our own fashion. I hope you will join MUUG in celebrating Christmas with our annual Christmas Wine and Cheese meeting on Tuesday, December 14, at 7:30 p.m. It will be held in the Atrium of the St. Boniface Hospital Research Center. In other words, the usual time, and just beside the usual place.

The Wine and Cheese meeting is always a great event. Everyone gets to relax and enjoy an evening of visiting with other members in a nice Christmas atmosphere. It's always a good opportunity to find out what everyone else is up to. It's also a chance for MUUG to say thanks to our members for their continued support. I'm serious about this point, but unfortunately it sounds like that wine cooler commercial.

Live Demos!

As another feature of our Christmas meeting, we will have demonstrations from a few different people. They will be showing off some of the interesting technologies that they are using and pursuing. Examples of what was at last year's demonstrations ranged from a live Linux demo, to a live music demo. Copperfield's bookstore also had a presentation of some of the many UNIX-related texts they have available.

Many of you must work with, or have as a hobby some interesting, and reasonably portable, technologies. If you'd like to demonstrate any of these, we'd love to have you. Please contact me to discuss how to enter your exhibit. You can email me at `bfinch@muug`, or call me at work, 934-2723. You don't have to build an elaborate presentation for this demonstration. Just setup whatever you have on a table in the atrium, and we'll even supply the table! Of course we'd need to know power requirements, etc., to get things ready ahead of time.

UnixWare?

Maybe someone will have a demo of UnixWare?? This was presented to us at our last meeting by Owen Sagness of Novell. I felt his presentation was one of the best we've had. He covered both the high level concepts, as well as fielding any technical questions given to him. All around, it gave us a good introduction into what UnixWare is, and where it's likely to go.

We will of course be continuing with our regular presentations again in January. Our first meeting topic will be UNIX security. The date for this meeting will be Tuesday, January 11, 1994. This should get us all back into business as usual after a good Christmas break.

And finally I'd like to take this opportunity to wish everyone all the best for the Christmas season. I hope you all have a happy holiday, and I'm looking forward to seeing you at the Wine and Cheese!

CORPORATE SPONSORS

The Manitoba UNIX User Group

gratefully acknowledges the generous support of the following

Corporate Sponsors

IBM

digital™

HEWLETT PACKARD

UM

XEROX The Document Company

Great-West Life Assurance Company

C++ Q&A

By Marshall P. Cline

The widespread acceptance and use of object-oriented methodologies has many people confused as to exactly what 'object-oriented techniques' are. The following monthly column will hopefully shed some light on the subject from the programmer's perspective. The C++ Q & A column was originally written by Dr. Mashall P. Cline (see bio at the end for more info), and is reprinted with permission. — ed.

Question 1: What is C++? What is OOP?

C++ can be used simply as 'a better C', but that is not its real advantage. C++ is an object-oriented programming language (OOPL). OOPLs appear to be the current 'top shelf' in the development of programming languages that can manage the complexity of large software systems.

Some OOP hype: software engineering is 'failing' to provide the current users demands for large, complex software systems. But this 'failure' is actually due to SE's *successes*. In other words, structured programming was developed to allow software engineers to design/build HUGE software systems (that's a success). When users saw how successful these systems were, they said, 'More — give me MOOORRRREEEE'. They wanted more power, more features, more flexibility. 100K line systems are almost commonplace nowadays, and they still want more. Structured programming techniques, some say, begin to break down around 100K lines (the complexity gives the design team too many headaches, and fixing one problem breaks 5 more, etc). So pragmatics demands a better paradigm than structured programming. Hence OO-design.

Question 2: What are some advantages of C++?

GROWTH OF C++ : C++ is by far the most popular OOPL. Knowing C++ is good for a resume, but don't just use it as a better C, or you won't be using all its power. Like any quality tool, C++ must be used the way it was designed to be used. The number of C++ users is doubling every 7.5 to 9 months. This exponential growth can't continue forever(!), but it is becoming a significant chunk of the programming market (it's already the dominant OOPL).

ENCAPSULATION : For those of you who aren't on a team constructing software mega-systems, what does C++ buy you? Here's a trivial example. Suppose you want a 'Foible' data type. One style of doing this in 'C' is to create a 'Foible.h' file that holds the 'public interface', then stick all the implementation into a 'Foible.c' file. Encapsulation (hiding the details) can be achieved by making all data elements in 'Foible.c' be 'static'. But that means you only get one 'Foible' in the entire system, which is ok if 'Foible' is a Screen or perhaps a HardDisk, but is lousy if Foible is a complex number or a line on the screen, etc. Read on to see how it's done in 'C' vs 'C++'.

MULTIPLE INSTANCES : The 'C' solution to the above 'multiple instances' problem is to wrap all the data members in a struct (like a Pascal 'record'), then pass these structs around as if they were the 'ComplexNumber' or whatever. But this loses encapsulation. Other techniques can be devised which allow both multiple instances and encapsulation, however these lose on other accounts (ex: typedef'ing 'Foible' to be 'void*' loses type safety, and wrapping a 'void*' in the Foible struct loses an extra layer of indirection). So the 'module' technique loses multiple instantiations, but the 'struct' technique loses encapsulation. C++ allows you to combine the best of both worlds - you can have what amount to structs whose data is hidden.

INLINE FUNCTION CALLS : The 'encapsulated C' solution above requires a function call to access even trivial fields of the data type (if you allowed direct access to the struct's fields, the underlying data structure would become virtually impossible to change since too many pieces of code would *rely* on it being the 'old' way). Function call overhead is small, but can add up. C++ provides a solution by allowing function calls to be expanded 'inline', so you have: the (1) safety of encapsulation, (2) convenience of multiple instances, (3) speed of direct access. Furthermore the parameter types of these inline functions are checked by the compiler, an improvement over C's #define macros.

OVERLOADING OPERATORS : For the 'Complex Number' example, you want to be able to use it in an expression 'just as if' it was a builtin type like int or float. C++ allows you to overload operators, so you can tell the compiler what it means for two complex numbers to be added, subtracted, multiplied, etc. This gives you: z0 = (z1 + z2) * z3 / z4; Furthermore you might want string1+string2 to mean string concatenation, etc. One of the goals of C++ is to make user defined types 'look like' builtin types. You can even have 'smart pointers', which means a pointer 'p' could actually be a user defined data type that 'points' to a disk record (for example). 'Dereferencing' such a pointer (ex: i=*p;) means "seek to the location on disk where p 'points' and return its value". Also statements like p->field=27; can store things on disk, etc. If later on you find you can fit the entire pointed-to data structure in memory, you just change the user-defined pseudo-pointer type and recompile. All the code that used these 'pseudo pointers' doesn't need to be changed at all.

INHERITANCE: We still have just scratched the surface. In fact, we haven't even gotten to the 'object-oriented' part yet! Suppose you have a Stack data type with operations push, pop, etc. Suppose you want an Invertable Stack, which is 'just like' Stack except it also has an 'invert' operation. In 'C' style, you'd have to either (1) modify the existing Stack module (trouble if 'Stack' is being used by others), or (2) copy Stack into another file and text edit that file (results in lots of code duplication, another chance to break something tricky in the Stack part of InvertableStack, and especially twice as much code to maintain). C++ provides a much cleaner solution: inheritance. You say 'InvertableStack inherits everything from Stack, and InvertableStack adds ☛

the invert operation'. Done. Stack itself remains 'closed' (untouched, unmodified), and InvertableStack doesn't duplicate the code for push/pop/etc.

POLYMORPHISM : The real power of OOP isn't just inheritance, but is the ability to pass an InvertableStack around as if it actually were a Stack. This is 'safe' since (in C++ at least) the is-a relation follows public inheritance (ie: a InvertableStack is-a Stack that can also invert itself). Polymorphism is easiest to understand from an example, so here's a 'classic': a graphical draw package might deal with Circles, Squares, Rectangles, general Polygons, and Lines. All of these are Shapes. Most of the draw package's functions need a 'Shape' parameter (as opposed to some particular kind of shape like Square). Ex: if a Shape is picked by a mouse, the Shape might get dragged across the screen and placed into a new location. Polymorphism allows the code to work correctly even if the compiler only knows that the parameter is a 'Shape' without knowing the exact kind of Shape it is. Furthermore suppose the 'pick_and_drag (Shape*) function just mentioned was compiled on Tuesday, and on Wednesday you decide to add the Hexagon shape. Strange as it sounds, pick_and_drag() will still work with Hexagons, even though the Hexagon didn't even exist when pick_and_drag() was compiled!! (it's not really 'amazing' once you understand how the C++ compiler does it — but it's still very convenient!)

***Dr. Marshall P. Cline** is the founder and President of Paradigm Shift, Inc., a firm that specializes in on-site training for C++, OOD, OOA, consulting, and reusable/ extensible C++ class libraries. For more information, send e-mail to "info@parashift.com".*

HANDS-ON

Gnu Review

By Peter Graham

In last month's column:
> Greetings everyone! If all goes well this will be a new regular
> column in your monthly newsletter. I am excited about writing
> this column and hope you will look forward to reading it.

Well, so far so good. I *just* made it this month so its been *regular* for two whole months now. :-)

Busy, Busy

Its exciting times at Pat's and my place. We just bought a used Sun system to use as a home machine and have been really busy getting it setup. It's a Sparc-1 with 17" grayscale monitor, 12MB of memory and 430MB of disk. Its not much compared to the hot new Sparc-10's or even Sparc-2's, but it meets our immediate needs. To make things busier, I am also trying to get a conference paper out to meet a December 1st deadline. Despite the rush, the arrival of the machine has given me an excellent excuse to grab and install a bunch of Gnu software.

Also in last month's column:
> I'll try to start out slow with simple software that all can make
> use of.

The best laid plans of mice and men.... Oh well!

gcc!

I had planned on reviewing a simple piece of software with applicability to the widest possible audience (I thought of doing "oleo" — the spreadsheet program) but the need for systems-oriented software on my workstation and the limited time frame dictates that I talk about something I just in-stalled. With this in mind, I thought I would do an about-face and get the biggest/ugliest install of all out of the way as quickly as possible. Yes, Virginia, I'm talking about the gcc/ g++ (C and C++) compiler. One advantage of talking about gcc now is that if you are going to install other gnu products they will often perform better if you compile them using gcc. Having some idea of what will happen during the gcc install may be of benefit to those who want install it and then compile other products with it.

Gcc is big! This is due in part to the fact that it is one of the FSF's "flagship" products (along with Gnu emacs) — it is an extremely good compiler that is portable to a huge number of machines and operating systems.

Installation Overview

Fortunately for those of you who are sysadmin neophytes, the install is not much more difficult than the install of any other Gnu product. There is a standard installation procedure for almost all Gnu software that consists of three basic steps: 1) running a configure script (./configure) — this script is pretty smart and on common machine/OS configurations does not require any arguments. It runs for a few minutes and figures out all kinds of things about your environment so that it can generate a customized Makefile for the next step. 2) run make (make) — this compiles the product using the customized Makefile produced by configure. For some of the larger products, gcc included, this actually involves more than one run of make. In the case of compiling gcc, three or four passes are required to generate a native mode compiler. (Generating a cross-compiler is quite a bit trickier but is well documented.) 3) run make again (make install) — this re-invokes the make file to install the required executables, libraries, and man pages. Gnu software is normally put in logical places by default (e.g. /usr/local/{bin,lib,man,...}). If you require other locations, it is easy to modify the Makefile(s) or you can provide different values for make variables (which control installation) on the make command line.

Due to the size of gcc, the installation is quite time consuming so be prepared to spend some time at it. On our Sparc-1 the total install took over 4 hours but only 5 or 10 minutes of this required user intervention. This means ☞

you can get other useful work done while installing gcc. (Assuming of course that you are doing the install on a multiprocess operating system like Unix and not on DOS, for example.)

Let's go over the installation process now.

Installation

First off, all Gnu software comes in compressed tar format (With the exception of products for the DOS environment.). The compression is no longer done with Unix compress (which has legal restrictions on it based on its use of the copyrighted/patented(?) Lempel-Ziv compression algorithm. Instead, the software is compressed using Gnu gzip and uncompressed using gnuzip (do not confuse these with the similarly named PC compression commands). These are available from the Internet Gnu archives (e.g. prep.ai.mit.edu) as, happily, uncompressed tar files.

Your first step is to ftp to prep.ai.mit.edu and cd to pub/gnu. Select binary transfer mode and issue a "get gzip-1.2.4.tar" command. This tar file is just a little under 800K bytes of data. Use the unix command "tar xvf gzip-1.2.4.tar" to extract the source code once you have copied the tar file to your local installation directory. You can now remove the tar'd version and use "cd" to go to the resulting directory (probably called "gzip-1.2.4"). Follow the directions in the README file which will consist of the basic steps outlined above. When you are done you will have a working version of gnuzip in an executables directory which is hopefully on your search path.

Now you're ready to fetch gcc. You can retrieve the compressed tar file from `gcc-2.5.4.tar.gz` in the same place you got gzip from. This file is about 6MB and extracts to over 25MB. Thus to install gcc you will need at least 50+MB of free space (25MB for the tar file and then 25MB for the extracted directories and files). (Egads!!!) Once you have extracted the directories and files using tar, be sure to get rid of the tar file itself since you will need the space for object files during the build process.

The 7 Steps to gcc

Finally, we can do the installation. Examining the README file that comes with the distribution we are horrified to find that it is 1625 lines long. We regain our composure after discovering that most of it is either notes for special machine configurations or part of a description of how to build cross-compilers. The basic installation consists of 6 or 7 steps as follows:

1) Run the configure - on my machine I didn't have to tell it anything about the environment. It figured it all out itself and did so without having to ask me any questions.
2) Issue the command "`make LANGUAGES=c`". Don't worry about specifying c++, etc. here since it isn't needed yet.
3) Issue the command "`make stage1`". This moves the result of the build of the gcc compiler in step 2 into a sub-directory named 'pass1'. This version of the compiler is then used to build the second, self-compiled, native mode compiler for C, C++, and/or Objective C.
4) Issue the command: "`make CC="stage1/xgcc -Bstage1/" CFLAGS="-g -O"`". This will build all three language compilers. If you want a subset, you can specify the make argument: "`LANGUAGES="LIST"`" where LIST should contain one or more of the words 'c', 'c++', and 'objective-c'. I built only C and C++ since I don't use Objective C. The times given below for installation reflect this fact.
5) Save some space and do a "`\rm -r stage1`".
6) At this point the compiler is built. You can test the compiler by having it compile itself by issuing the commands "`make stage2`" and "`make CC="stage2/xgcc -Bstage2/" CFLAGS="-g -O"`" followed by "`make compare`". I omitted this step.
7) You now install the compiler using: "`make install CC="stage2/xgcc -Bstage2/" CFLAGS="-g -O" LANGUAGES="LIST"`" where list should be the same as that given in step 4.

Almost Done

Congratulations! You have successfully installed gcc, etc. Almost! If you chose to compile the g++ compiler as well, you have a fully functional C++ compiler but without any class libraries. These are distributed separately but may also be picked up at `prep.ai.mit.edu: /pub/gnu/libg++-2.5.1.tar.gz`. This archive is about 1.3MBs and installs in much the same way (only faster) as gcc.

For each product reviewed, I will provide a quick overview in the following format which I use here to summarize gcc/g++.

gcc Summary

Name	gcc
Description	C, C++, and Objective C compiler.
Archive Loc'n	prep.ai.mit.edu: /pub/gnu/gcc-2.5.4.tar.gz
Archive size	5984168 bytes
Approx Space to Install	50+MB Time to Install (Sparc-1): 4.5 hours (10 minutes interactive)
Pros	Excellent compiler for 3 languages. Fast executable code (many optimizations) Highly Portable Conforms to K&R, ANSI, C++ V2 and V3 Many Extensions/Improvements Free Software
Cons	Compiles slightly slower due to optimizations (may be made faster by turning opt's off.) No free support (companies offer support for a price if you want it). Needs lots of space/time to install.

Next Month

That's it for this month. Next month we will look at another Gnu product. If you have suggestions for additional information you would like to see in the reviews, feel free to email me at `pgraham@cs.umanitoba.ca` and I will try to be accommodating.

New "Politically Correct" UNIX

By Ernest Prabhakar

Submitted By Andrew Trauzzi

Politically Correct UNIX
System VI Release notes

In order for UNIX to survive the rest of this decade, it must get rid of its intimidating commands and outmoded jargon, and become compatible with the existing standards of our day. To this end, our technicians have come up with a new version of UNIX, System VI, for use by the PC — that is, the "Politically Correct."

Utilities

- "man" pages are now called "person" pages.
- Similarly, "hangman" is now the "person_executed_by_an_oppressive_regime."
- To avoid casting aspersions on our feline friends, the "cat" command is now merely "domestic_quadruped."
- To date, there has only been a UNIX command for "yes" — reflecting the male belief that women always mean yes, even when they say no. To address this imbalance, System VI adds a "no" command, along with a "-f[orce]" option which will crash the entire system if the "no" is ignored.
- The bias of the "mail" command is obvious, and it has been replaced by the more neutral "gender" command.
- The "touch" command has been removed from the standard distribution due to its inappropriate use by managers.
- "compress" has been replaced by the lightweight "feather" command. Thus, old information (such as that from Dead White European Males) should be archived via "tar" and "feather".
- The "more" command reflects the materialistic philosophy of the Reagan era. System VI uses the environmentally preferable "less" command.
- The biodegradable "KleeNeX" displaces the environmentally unfriendly "LaTeX".

Shell Commands

- To avoid unpleasant, medieval connotations, the "kill" command has been renamed "euthanise."
- The "nice" command was historically used by privileged users to give themselves priority over unprivileged ones, by telling them to be "nice". In System VI, the "sue" command is used by unprivileged users to get for themselves the rights enjoyed by privileged ones.
- "history" has been completely rewritten, and is now called "herstory."
- "quota" can now specify minimum as well as maximum usage, and will be strictly enforced.
- The "abort()" function is now called "choice()."

Terminology

- From now on, "rich text" will be more accurately referred to as "exploitive capitalist text".
- The term "daemons" is a Judeo-Christian pejorative. Such processes will now be known as "spiritual guides."
- There will no longer be a invidious distinction between "dumb" and "smart" terminals. All terminals are equally valuable.
- Traditionally, "normal video" (as opposed to "reverse video") was white on black. This implicitly condoned European colonialism, particularly with respect to people of African descent. UNIX System VI now uses "regressive video" to refer to white on black, while "progressive video" can be any color at all over a white background.
- For far too long, power has been concentrated in the hands of "root" and his "wheel" oligarchy. We have instituted a dictatorship of the users. All system administration functions will be handled by the People's Committee for Democratically Organizing the System (PC-DOS).
- No longer will it be permissible for files and processes to be "owned" by users. All files and processes will own themselves, and decided how (or whether) to respond to requests from users. The X Window System will hence forth be known as the NC-17 Window System.

And finally, UNIX itself will be renamed "PC" — for Procreatively Challenged.

CORPORATE MEMBERSHIPS

Are there other people at your office who wish to join
The Manitoba UNIX User Group?

Well now is a better time than ever before with
MUUG Corporate Memberships!

- **Ten employee minimum**
- **Memberships are twenty dollars each**
- **Two members will have voting rights**
- **Corporate MONA access is not available, but**
- **Personal MONA accounts are available**

For details, contact any MUUG board member.

SIG Sideline

By Bary Finch, SIG Coordinator

Our last SIG meeting had the usual good turnout of around 15 people. This time we indulged in the round table format of discussion, i.e. “talk techie”. There was no specific topic presented.

Of course, many different topics came up through the night. There was quite a bit of “X” based talk, with some discussing the X11 implementation in Linux. Others got quite involved in discussing many of the DOS X-window packages that are available. Many places are using these packages as a way of integrating X-terminal function into their PC environment. People were discussing the benefits of one package over another, and some specific problems that they had experienced. It was a good introduction to anyone who was new to the idea of running “X” from a PC to a UNIX machine.

Another interesting point was that MS Windows is starting to be written for Linux. It’s estimated it will be well into 1994 before anything is really delivered, but it’s under way. It won’t be a version of WABI (Windows Application Binary Interface) from SUN, but it will get your MS Windows applications going.

As for our next meeting, please note:

DECEMBER 21ST SIG MEETING IS CANCELLED
(due to it being on Christmas week)

Sort of sounds like the old joke line of “Christmas is cancelled due to . . . ”, but quite the opposite. We didn’t get any response from a news group posting asking for interest, and the people that I know of will not be available that week due to many Christmas commitments. So we will let everyone indulge fully in the Christmas spirit(s), and skip December.

Our next meeting will then be held on Tuesday, January 18, at 7:30 p.m. We intend to have another presentation at this meeting. No specific topic has yet been finalized, but it will likely be from the original list of topics generated at the first SIG meeting.

The desired topics determined were: networking, working with users, installing LINUX, mail, x-windows, and setting up peripherals like terminals, printers, and modems. This all culminates in a major interest expressed in learning how to connect to MONA to poll for mail.

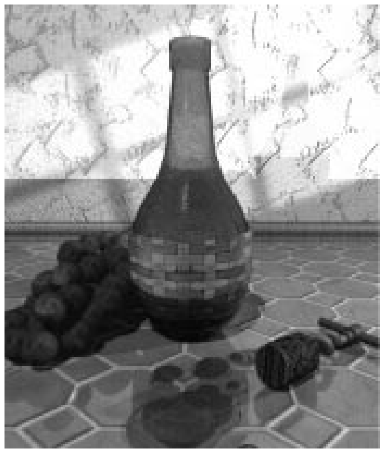
So we will continue to work towards the final goal above. If anyone is interested in being a guest speaker at a SIG meeting on one of the listed topics, let me know. Or if you have a specific topic of interest, let me know that too. As usual, my email is `bfinch@muug`, and my work phone is 934-2723.

That’s about it for this month. I hope you all get the best out of the Christmas season, and enjoy a Happy New Year! We’ll see you at the January SIG meeting!

Agenda

for
Tuesday, December 14, 1993, 7:30 PM
Samuel N. Cohen Auditorium
St-Boniface Hospital Research Centre
Main Floor, 351 Taché

The MUUG Annual Wine and Cheese Party



Held in the atrium of the St. Boniface Hospital Research Centre (outside the lecture theatre). Don’t miss it!

Coming Up

Meeting:
Next month’s meeting is scheduled for Tuesday, January 11, at 7:30 PM. Meeting location will be the St-Boniface Research Centre, as usual. The January meeting topic is UNIX Security. Stay tuned for details.

Got any ideas for meeting topics? Any particular speaker, company, or product you’d like to see at one of our meetings? Just let our new meeting coordinator, Roland Schneider, know. You can e-mail him at <rsch@muug.mb.ca>.

Newsletter:
If you are interested in a particular topic, let me know. I’m sure I could coerce you into writing an article! I could use a few articles — especially shorter ones — half a page to one page (400 to 1000 words) would be fine.

Monsieur Ex has also let me know that his mail-box has room for more of your wonderful queries again – please submit your questions to the old guy via e-mail to <m-ex@muug.mb.ca>. He may be old, but he’s not ready for retirement yet!