November 1989 $2.50
Volume 2, Number 2

Technical UNIX® User Group

newsletter of the Technical UNIX® User Group

This month ...

Late Breaking News...
Next Meeting to be held at UNISYS
See inside for details

Thoughts From The Editor

By Susan Zuk

It's hard to believe that we have started a new volume of the newsletter. The past year has passed so quickly!

For those that don't know me, my name is Susan Zuk, and I am with the company, UNISYS. Last year I was secretary but I have relinquished that role to Matt Binnie. I am very thankful to last year's editor, Darren Basler, for the great job he did in designing and putting the newsletter together. It has saved me a tremendous amount of time being able to use his ground work. If you have any ideas or articles for the newsletter just give me a call. My phone number is listed below.

One reason for UNIX becoming so popular is its versatility. Developers wrote it to have the ability to perform things quickly and with the least amount of typing, as you can see from the short command names like *ls* for list and *cp* for copy.

The November Newsletter contains a couple of very interesting articles. Gilles Detillieux's discussion on printer spooling will give you some hints on how to customize the way print jobs are sent to your printer and also will help to start you creating your own spooler interface files for that special printer you want to use. Gilles also included a sample file which he uses to explain the spooling procedure.

The second article comes to us from /usr/group's CommUNIXations Magazine. This article, on Bourne shell functions, provides ideas for creating either your own shell commands or adding your favorite options to a current command such as *ls*.

If you have performed some customization work on your own computer, write it up and let us share it with the rest of the group.

In the final portion of the Newsletter, we have listed our Financial Statement for the past season. We are ending the year in the black. Maybe we should give the government some tips!

Anyway, I shall say bye for now and let you enjoy this month's TUUG Newsletter.

Group Information

The Technical Unix User Group meets at 7:30 pm the second Tuesday of every month, except July and August. The newsletter is mailed to all paid up members 1 week prior to the meeting. Membership dues are $20 annually and are due at the October meeting. Membership dues are accepted by mail and dues for new members will be pro-rated accordingly.

The Executive

President:	Gilbert Detillieux	261-9146
Vice President:	Derek Hay	943-5401
Treasurer:	Gilles Detillieux	261-9146
Secretary:	Matt Binnie	(W) 949-0190
Newsletter Editor:	Susan Zuk	(W) 788-7312
Membership Sec.:	Pat Macdonald	(W) 474-9870
Information:	Gilbert Detillieux	261-9146
	(or) Susan Zuk	(W) 788-7312

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

Copyright Policy and Disclaimer

ANNOUNCEMENT...

Meeting Location Change:
The November meeting location will be provided by UNISYS Canada Inc., Suite 300-1661 Portage Ave (UNISYS Building). Upon entering the building you will then be required to sign-in. Please sign-in using "TUUG" as the agency represented.

President's Corner

by Gilbert Detillieux, President

As I'm sitting down to write this, on October 17, the big news story is the earthquake that just rocked San Francisco. Rather appropriate, since I feel like I'm in the middle of one as I rush to get things done in time. The newsletter deadline has been moved up, since I will be out of town for a couple weeks starting on October 25th. In addition to rushing to finish this article, I've had to write a program for our new editor, Susan, to help prepare text files containing articles submitted for inclusion in the newsletter, and also handle our company's year end finances, and start work on a new job. You'll have to excuse me if this article seems a little schizophrenic.

In case you missed the last meeting, you may be interested to know that the new executive was elected by acclamation. The list of candidates shown in my column last month now makes up the executive for the 1989-1990 year, ending next September. I would like to take this opportunity to welcome the two new members of the executive, Derek Hay (Vice-President) and Matt Binnie (Secretary), and also thank them for volunteering some of their time to help out the group. I would also like to thank our out-going editor, Darren Besler, for all the time and effort he put into the newsletter over the past year. We'll miss you, Darren!

The executive will have had its first meeting by the time you read this, and hopefully will have lots of things planned for the upcoming months. The December meeting will likely be a small pre-Christmas party, similar to last year. We also have a lot of ideas for topics and speakers for meetings in the following months; we'll let you know as soon as we can confirm some of these. Also, if you have any ideas or wish-lists for topics and/or speakers, please bring them to the attention of any member of the executive.

It looks like the November 14th meeting will be held at Unisys again. The presented topic, after the business meeting, will be a surprise. In other words, even I don't know what the topic will be yet - one possibility is a presentation on DOS/UNIX interfacing. Look for updates elsewhere in the newsletter.

Well, this is about all I have time for, boys and girls (Sorry, Susan, looks like you'll need some filler), 'cause I've got lots of work left to do. So I'll see you all at the next meeting, or talk to you in the next newsletter. Bye for now, and watch out for those little ghosts and goblins!

The fortune file

This month's fortune comes courtesy of Gilbert Detillieux, who found it on a DOS program called MURPHY.

```
Harrison's      Postulate:

   "  For   every   action,   there   is   an   equal   and   opposite   critisism."
```

The 1989-1990 Executive

President:	Gilbert Detillieux, InfoWest
Vice President:	Derek Hay, FACS Records Center
Treasurer:	Gilles Detillieux, InfoWest
Newsletter Editor:	Susan Zuk, UNISYS
Membership Secretary:	Pat Mcdonald, University of Manitoba
Secretary:	Matt Binnie, PRIME
Meeting Coordinator:	Matt Binnie, PRIME

An Improved Interface Model for the LP

by Gilbert Detillieux, Info West Inc.

The LP spooler, that comes with most UNIX systems today, is a big improvement over early UNIX spoolers, such as LPR. In addition to supporting multiple printers, and providing more control over spool queues than earlier spoolers, LP is a big improvement because it can readily be configured to support different types of printers. This configurability is provided by allowing you to select one of several printer ''interface'' programs, or by defining your own.These programs are simply shell scripts, which can easily be customized.

In addition to allowing you to support different types of printers, these interface programs also allow you to add your own local options to the LP spooler, to control the operation of the printer. This capability is a powerful way of enhancing the versatility of the spooler. In fact, many of the limitations of LP, that often frustrate novice users (such as the banner page always being printed, and blank pages between pre-paginated files), are not inherent limitations in LP, but simply due to inadequate interface programs.

This article presents a model for a more flexible LP interface that provides local options to suppress the banner page, suppress form feeds after each file printed, and to control printing of raster graphics. This interface is for an HP PCL printer, such as the LaserJet, but is general enough to be used with any type of line printer. In fact, the only printer specific component is a separate filter program to convert a raster graphic image file into PCL raster graphics commands. This filter, which is also specific to a particular graphic file format, is beyond the scope of this article, and is therefore not listed. The interface program itself, a shell script, is presented in Listing 1.

How It's Used.

In its default mode of operation, it works just like the standard LP interfaces. It starts by printing a banner page, containing the user's login name and an optional title in large type. It then prints the text of each file, with a form feed after each, and repeats this for the requested number of copies. This behavior can be altered by specifying local options to LP.

If the ''nb'' (no banner) or ''nh'' (no header) option is specified, for example:

```
lp -onb sample.txt
```

the banner page will be suppressed. This may be desirable especially when printing on expensive paper or pre-printed forms.

If the text to be printed is already paginated, such as the output of the ''pr'' command, the ''nf'' (no form feed) option can be used to suppress the form feed that is normally done after each file:

```
pr sample.txt | lp -onb -onf
```

Other options are provided to indicate how files are to be processed, allowing for printing of graphics in addition to text. By default, or if the ''text'' option is explicitly specified, each file argument is treated as a normal UNIX text file to be printed. This means that the files' contents are sent to the printer as is, except that newlines are translated to CR/LF sequences on output. (For a PostScript printer interface, a filter program would be needed to translate lines of text to the PostScript commands to print that text.)

If the ''raw'' option is specified, each file is sent to the printer in raw mode, i.e. without any post-processing. The files would likely be binary files containing printer-ready data. This could be downloadable fonts, setup commands, formated text orgraphics, for example.

If the ''graph'' option is specified, each file is assumed to contain a raster graphic image, and is passed to a filter program to convert the graphic to the format required by the printer. If there are several different graphic file types to be handled, different options could be interpreted by the interface, and passed on to the appropriate filter program, or the filter program could be designed to automatically recognize the different formats. The interface in this example also recognizes options to set the rotation, resolution, and inversion of the graphic, and pass them on to the filter. Other such options could be added, as appropriate.

This interface could also be easily enhanced to support other types of files, such as HPGL plot files, for example. An ''hpgl''option could be recognized to handle such files. For an interface to an HPGL compatible plotter, this would be treated like the ''raw'' option. For other types of printers and plotters, an appropriate filter could be called to convert the HPGL code to graphic commands for that device.

How It Works

The interface program, once installed for a particular LP printer, is called by ''lpsched'' for each print request to that printer, with all required information and options passed as arguments. The interface first sets up a reasonable set of defaults, then scans its fifth argument, the option list. Each specified option is processed in turn, and causes certain variables to be set to values that will alter the default behaviour. If an unrecognized option is encountered, the interface program ignores the entire print request, and mails an error message to the issuing user.

Once all options have been handled, the program sets up the output port as appropriate for printing text files. Next, it goes on to print the banner page, if one is desired (i.e. no "nb" or "nh" option was given). This banner page is generated in the same way as for the standard LP interfaces.

Next, if binary data (as opposed to text files) are to be printed, the output port is set up to disable outputpost-processing. Each file is then handled in turn, for the requested number of copies. Files will be processed according to any printing mode options (such as "graph" or "raw") given, and the appropriate filter program will be called to convert the file, or the "cat" command will be used if no conversion is needed. Form feeds will be printed after each file, unless an "nf" option was given.

By adding new options to this model, and adding the appropriate filter programs, the interface could be adapted to handle any type of printer you use, and any type of files you want printed. Naturally, it could take a fair bit of programming to develop the required filters, but it's nice to have that sort of power and flexibility in a system, in case you need it.

****

*Listing 1: Example of an LP interface for an HP PCL printer

```
#! /bin/sh
# hppcl - lp interface for HP PCL printers, eg. LaserJet.
#
# (c) 1988, 1989, INFO WEST Inc.  All rights reserved.
#
# called with args: printername req-id username title #copies options file ...
# for parameters: $0        $1    $2      $3    $4      $5     $6 ...
# options are:
#        nh or nb - no header (banner) page is generated
#        nf             - no form feed sent between files, or at end
#        text           - treat files as UNIX text files with newlines (default)
#        raw            - print files in raw mode (8 bits, no opost)
#                         (could be printer-ready graphics, fonts, or text)
#        graph[ics]     - treat files as raster graphics to be printed
#        rot=deg        - print raster graphics with given rotation
#                               (rounded to nearest multiple of 90)
#        res=dpi        - print raster graphics at given resolution
#        inv=x          - if x is non-zero, invert bits for printing,
#                         otherwise, treat 0 as black

# Set up default parameters, for printing text files:
devmode='9600 cs8 -parenb ixon -ixoff istrip icanon -hupcl opost onlcr -tabs'
prmode=text             # default printing mode
banner=yes              # generate banner page
formfeed='\014'         # formfeed string to be echo'ed

# Setup for optional graphics filter program:
rasfilter=/usr/lib/hppclras # filter to print raster graphics
rot=-90                 # default raster orientation
res=150                 # default raster resolution
inv=1                   # default inversion parameter (treat 0's as white)

# Scan options argument:
for option in $5
do
        case "$option" in
        nh|nb)  banner=no; continue ;;          # no banner page
        nf)     formfeed= ; continue ;;         # no formfeed
        text|raw)                       # set printing mode
                prmode=$option; continue ;;
        graph*)                                 # set graphics mode, if possible
                if [ -x $rasfilter ]    # i.e. if program exists
                then
                        prmode=$option; continue
                fi;;
        rot=*|res=*|inv=*)                      # set graphics option
                eval $option; continue ;;
        esac
        # if we reach here, invalid option encountered
        # so ignore request, and mail error message to user
        prname=`basename $0`
        mail $2 <<MAILMSG
```

```
Request id: $1     Printer: $prname
Title: $3    Copies: $4
Options:  $5
This printer interface was called with invalid/unsupported option: $option
Print request ignored.
MAILMSG
        exit 0
done

# Set printer port mode as appropriate for text files:
stty $devmode <&1

# Generate banner page, if required:
case $banner in
yes)

x="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
        echo  "$x\n$x\n$x\n$x\n"
        banner "$2"
        echo "\n"
        user=`grep "^$2:" /etc/passwd | line | cut -d: -f5`
        if [ -n "$user" ]
        then
                echo "User: $user\n"
        else
                echo "\n"          fi
        echo "Request id: $1     Printer: `basename $0`\n"
        date
        echo "\n"
        if [ -n "$3" ]
        then
                banner  $3
        fi
        echo  "\n\n$x\n$x\n$x\n$x\n"
        echo "\014\c"
        ;;
esac

# Process file arguments:
copies=$4
shift; shift; shift; shift; shift
files="$*"

# If printing binary files (not text), set mode:
case $prmode in
text)     ;;
*)        stty -opost <&1 ;;
esac

case $prmode in
text|raw)          # Process non-graphic files
        i=1
        while [ $i -le $copies ]
        do
                for file in $files
                do
                        cat "$file" 2>&1
                        echo "$formfeed\c"
                done
                i=`expr $i + 1`
        done
        ;;
*)      # Process raster graphics file(s):
        i=1
        while [ $i -le $copies ]
        do
                for file in $files
                do
                        $rasfilter -rot $rot -res $res -inv $inv $file
                        echo "$formfeed\c"
                done
                i=`expr $i + 1`
        done
        ;;
esac
exit 0
```

Solving Those Puzzling Quirks of UNIX Systems Use

By Steven List and Bruce Syewart

Reprinted from /usr/group CommUNIXations May/June 1988

With the advent of UNIX System V, the Bourne shell added a new and powerful capability: functions. Shell functions are exactly what the name suggests - modular, reusable procedures to perform a specific task. Once written, shell functions may be used by any shell program that needs their services, including the login shell. Users can build collections of useful shell procedures or customize their environment by creating their own commands.

Shell functions offer all the advantages of C functions and others as well. Because shell functions accept parameters, their actions can be tailored to the purpose of calling the script or function. In addition, shell functions can use environmental variables directly, simply be referencing them. Finally, shell functions execute quickly; once defined to a script (and here we include the login shell as well), shell functions execute from memory and do not have to be read in from disk. This feature makes shell functions the method of choice for creating custom shell commands.

Defining Shell Functions

Figures 1a and 1b illustrate two forms of a shell function definition. Figure 1a shows the full form of the function definition. In this form, the function name is followed by an empty pair of parentheses. On the next line is an opening brace, signalling the beginning of the function body. The body of the function can reference shell commands, other shell scripts or shell functions. The (optional) return statement completes the body, followed by a closing brace to mark the end of the function definition.

Figure 1b shows the same function definition in a single line. Note that a semicolon appears before the closing brace. The semicolon is required for one-line function definitions.

Arguments are passed to shell functions as positional parameters ($1, $2, etc.), exactly as they are in the Bourne shell. Arguments to shell functions are independent of those passed to the parent shell. Within a shell function, $3 refers to the third argument passed to the shell function, not to the third argument passed to the parent shell. Shell functions can also read and test environment variables directly.

Customizing Your Environment

Shell functions are particularly useful for customizing your normal working environment. Rather than writing a specialized shell script to create a new command, consider using a shell function. For example, suppose you often need to look for hidden files and want to sort them by date of last access. You could create a shell function for this task.

```
dotfiles()
{
    for i in 'ls -aut | fgrep '^\.'`
    do
        if [ -f $i]
        then
            ls -laut $i
        fi
    done
}
```

This translates to: *List all files in the current directory, including hidden files. Sort the list by last access date, and select only*

Figure 1a

```
funcname()
{
        body
        return retval
}
```

Figure 1b

```
........... funcname() { body;}
```

those files whose names begin with a dot. For each one of these, show a long listing with time of last access.

Note that an extra step was needed to restrict the output to files (not directories) whose names begin with a dot. Without this step, the current directory (dot) and the parent directory (dot-dot) both qualify, and all files in both directories are listed.

The shell function lists all of the hidden files in the current directory sorted by date of last access. With a few changes, you can use the fuction to list the hidden files in any directory, not just the current one:

```
dotfiles()
{
    for i in 'ls -aut $1 |grep '^.'`
    do
        if [ -f $1/$i ]
        then
            ls -laut $1/$i
        fi
    done
}
```

The function now expects a directory name to be passed as an argument. This directory is then searched for hidden files, and the resulting listing is sorted by date of last access.

You could use this function to create another shell function to search as many directories as you like for hidden files. The new function might look like this:

```
listdot()
{
    for i in $*
    do
        echo $i:
        dotfiles $i
        echo
    done
}
```

You now have two functions: *dotfiles*, to display the hidden files in a single directory; and *listdot*, to invoke *dotfiles* once for each directory passed as an argument. You can combine these two functions into a single file; they become available as soon as you define them to the shell with the . (dot) command, the Bourne shell equivalent of *source* in the C shell.

For example, if you put both of these functions in a file named dotcheck, you would use the command to define the functions to the shell:

```
".dotcheck"
```

The *set* command can be used to verify that the functions have been defined to the shell because shell function definitions will appear in the output of the command.

With these two functions, you can produce a report of all the hidden files in each directory on the system. Within each directory, the hidden files will be sorted by date of last access. To produce the report, you can use the command:

```
listdot `find / -type dir -print`
```

This invokes *dotfiles* for every directory on the system. Of course, not all of these directories will be readable (unless you have super-user priviledges).

Lazy or Efficient?

Shell functions can also be used to create aliases or shortened forms of commands in the Bourne shell. Some users like to abbreviate their most frequently used commands with single letters, even when this saves only two or three keystrokes. Other users prefer to create shorter, easily remembered abbreviations for long command lines, or a command with lengthy options. However, there are pitfalls here, too. A shell function such as:

```
ls()
{
    ls -aFC
}
```

will not map the standard *ls* command to one of your favorite options. The definition is circular; although the shell will accept such a function definition, the results will probably not be what you intended. Use the following format to make your favorite options the default on the *ls* command:

```
ls()
{
    /bin/ls -aFC
}
```

In Conclusion

Shell functions are powerful tools that can make life in the Bourne shell much more enjoyable. They can be used to abbreviate lengthy or complicated commands, much like aliases in the C shell. They are efficient because they execute from memory and do not have to be read in from disk at each invocation. And most importantly, they allow users to create modular, flexible procedures that can be called from other shell scripts or functions, including the login shell.

Steven List is cofounder of Transact Software Inc., (TSI), developer of custom applications for clients in the UNIX environment. Bruce Stewart is an independent software consultant.

Technical UNIX User Group Financial Statements

Gilles Detillieux, Treasurer

Balance Sheet

		Mar '89	Sep '89
Assets: bank account		324.83	81.62
Liabilities:			
accounts payable			
	- postage	75.26	
	- paper	22.50	—
	Total	97.76	0.00
Equity: net income		227.07	81.62
Total liabilities + equity		324.83	81.62

Income and Expenses

		Mar '89	Sep '89*
Income: membership dues		478.00	546.00
Expenses:			
barbeque		0.00	49.87
bank service charges		4.05	5.50
cheque printing		32.11	32.11
name search + notation		30.00	30.00
postage		75.26	232.97
stationery			
	- envelopes	33.17	33.17
	- rubber stamp	19.87	19.87
	- mailing labels	33.97	33.97
	- paper	22.50	26.92
	Total	109.51	113.93
Total expenses		250.93	464.38
Net income		227.07	81.62

* Sep '89 income and expenses are for whole year (Oct - Sep)

Technical UNIX® User Group

Minutes From the Business Meeting
~~September 12~~, 1989

Oct. 10

1. Minutes:

   MOTION : (Gilles Detillieux) The minutes from the September 12th, 1989 meeting be approved.

   SECONDED : (Darrin Besler)

In Favour : 13	Opposed : 0	Carried

2. Membership Report:

   Please fill out the membership forms you received with the newsletter. The October newsletter is the final newsletter of this year.

3. Newsletter Report:

   Please contribute articles for the newsletter.

4. Treasurers Report:

   The financial report will be inserted into the November newsletter.

5. 1989-1990 Executive Nomiations:

   President : Gilbert Detillieux
   Vice President: Derek Hay
   Treasurer:Richard Willacy
   Treasurer: Gilles Detillieux
   Newsletter Editor: Susan Zuk
   Newsletter Editor: Darren Besler
   Secretary: Matt Binnie
   Meeting Co-ordinator: Matt Binnie

   Motion: (Kirk Marat) Let the above mentioned members be accepted as the Executive for th 1989/90 term.

   Seconded: (Darren Sampson)

Infavour: 13	Opposed: 0	Carried

Technical UNIX® User Group

Agenda

for

Tuesday, November 14, 1989

7:30pm

UNISYS

UNISYS Building

300-1661 Portage Avenue

1. Round Table	7:30
2. Business Meeting	8:00
a) Minutes of September's Meeting	
b) Membership Secretary's Report	
c) Newsletter Report	
d) Treasurer's Report	
4. Break	8:30
5. Presented Topic	8:40
Surprise !!!	
6. Adjourn	9:30